1-1-2008

Automated grazing management

Gregory J. Von Pless

Automated Grazing Management
Master's Thesis

Gregory J. Von Pless
gjv2098@cs.rit.edu
Department of Computer Science
Rochester Institute of Technology

May 16, 2008

Signatures

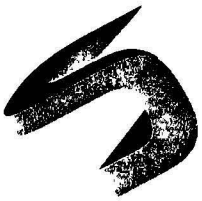
I, Gregory J. Von Pless, do hereby submit this thesis in partial fulfillment of the requirements for the degree of Master of Science in Computer Science. It is approved by the committee members below.

Gregory J. Von Pless

Dr. Zack Butler, Ph.D.
Committee Chair

Dr. Leon Reznik, Ph.D.
Reader

Dr. Joe Geigel, Ph.D.
Observer



Thesis/Dissertation
Author Permission Form
Graduate Student Form

Rochester Institute of Technology
B. Thomas Golisano College of
Computing and Information Sciences
Department of Computer Science
Building 70, Room 3005

Title of Thesis or Dissertation Automated Grazing Management

Name of Author Gregory J. Von Pless

Degree MS

Program Computer Science

College GCCIS

I understand that I must submit a print copy of my thesis or dissertation to the RIT Archives, per current RIT guidelines for the completion of my degree. I hereby grant to the Rochester Institute of Technology and its agents the non-exclusive license to archive and make accessible my thesis or dissertation in whole or in part in all forms of media in perpetuity. I retain all other ownership rights to the copyright of the thesis or dissertation. I also retain the right to use in future works (such as articles or books) all or part of thesis or dissertation.

Print Reproduction Permission Granted

I, Gregory Von Pless, hereby **grant permission** to the Rochester Institute of Technology to reproduce my print thesis or dissertation in whole or in part. Any reproduction will not be for commercial use or profit.

Signature of Author _____ Date 5/15/08

Print Reproduction Permission Denied

I, _____, hereby **deny permission** to the RIT Library of the Rochester Institute of Technology to reproduce my print thesis or dissertation in whole or in part.

Signature of Author _____ Date _____

Inclusion in RIT Digital Media Library ETD Archive

I, Gregory Von Pless, additionally grant to the Rochester Institute of Technology Digital Media Library (RIT DML) the non-exclusive license to archive and provide electronic access to my thesis or dissertation in whole or in part in all forms of media in perpetuity. I understand that my work, in addition to its bibliographic record and abstract, will be available to the world-wide community of scholars and researchers through the RIT DML. I retain all other ownership rights to the copyright of the thesis or dissertation. I also retain the right to use in future works (such as articles or books) all or part of this thesis or dissertation. I am aware that the Rochester Institute of Technology does not require registration of copyright for Electronic Thesis and Dissertations (ETD). I hereby certify that, if appropriate, I have obtained and attached written permission statements from the owners of each third party copyrighted matter to be included in my thesis or dissertation. I certify that the version I submitted is the same as that approved by my committee.

Signature of Author _____ Date 5/15/08

Acknowledgments

I would like to thank all of my family members and friends who were subjected to talk of cattle and grazing management at great length throughout this process — especial thanks to my parents, and to Chris, Liz, Tino, and Tay, who I believe suffered the lion's share in this regard. I'd like to thank Professor Butler for his guidance and continued patience as this process dragged on.

Thanks to Linus and Sam and their graduate student minions for putting up with my myriad requests for runaway processes to be killed so my simulations could finish this century. Finally, a special thank you to Tina Sturgis, who has given me aid and advice on countless occasions, and who helped me convince myself to get into this whole Master's degree mess in the first place.

Contents

1 Introduction **1**

2 Related Work **4**
2.1 Computerized Animal Control 4
2.1.1 Herding With Robots 4
2.1.2 Dynamic Virtual Fences 5
2.2 Evolutionary Algorithms and Expert Systems 6
2.2.1 Evolutionary Algorithms 6
2.2.2 Expert Systems 7
2.2.3 Optimizing Expert System Parameters 8
2.2.4 Building Expert Systems 8
2.2.5 A Hybrid Hybrid Approach 13
2.2.6 Adaptive Expert Systems 14

3 Algorithm Design **16**
3.1 Herding Algorithm 16
3.1.1 General Procedure 17
3.1.2 Variables 18
3.1.3 Adaptive Expert System 19
3.2 Evolutionary Algorithm 22
3.2.1 Genotype 23
3.2.2 Crossover 23
3.2.3 Mutation 24
3.2.4 Parameters 24
3.2.5 Fitness Calculation 25
3.2.6 Initial Population 27

4 Implementation and Evaluation **28**
4.1 Simulator 28

4.1.1 Forces 29
4.1.2 Vegetation Consumption and Growth 32
4.2 Experiments 32
4.2.1 Herd Drift 35
4.2.2 Vegetation Quality and Elevation 37
4.2.3 Full Simulation 40
4.2.4 Adaptive Expert System 45
4.2.5 Reduced Thirst Frequency 48
4.3 Results Analysis 50

5 Conclusion **53**

References **54**

List of Figures

3.1 Grazing management algorithm 17
3.2 On-line adaptation algorithm 21
3.3 Defining membership functions 23

4.1 Seed system 1 memberships 33
4.2 Seed system 2 memberships 34
4.3 Seed system 4 memberships 35
4.4 Experiment 1 field 36
4.5 Experiment 1 results 37
4.6 Experiment 2 field 38
4.7 Experiment 2 control results 38
4.8 Experiment 2 results 39
4.9 Experiment 3 field 41
4.10 Experiment 3 control results 41
4.11 Experiment 3 results 43
4.12 Experiment 3 stress-based fitness EVENNESS memberships . . 44
4.13 Experiment 3 combined fitness HAPPINESS memberships . . . 45
4.14 Experiment 4 results 46
4.15 Experiment 4 result field 2 explained 49
4.16 Experiment 4 control and results 50

List of Tables

3.1 Expert system operators 20
3.2 Evolutionary algorithm parameters 25

Abstract

Management-intensive grazing is a grazing management strategy that relies on careful monitoring of animals at pasture and frequent relocation of animals between various regions of the pasture, or paddocks, in order to maximize the nutrients and sustenance the animals obtain through grazing. When applied successfully, this approach to grazing management yields higher animal production, while cutting feed costs; however, a great deal of overhead is introduced in monitoring and moving the animals throughout the pasture, making this approach very difficult to implement at a large scale.

Recent successes in the field of dynamic virtual fencing have demonstrated the feasibility of automatically restraining and moving cattle within a pasture, and without the need to build or move fences. This technology may be used to reduce the physical overhead of management-intensive grazing, but it does not address the decision-making aspects.

This thesis proposes a decision-making system to be used in conjunction with dynamic virtual fence technology to implement a fully-automated intensive grazing management strategy. The system has been implemented and tested in simulation, and the results thereof are presented and analyzed.

Chapter 1

Introduction

Grazing management is the process through which animal foraging behaviors — specifically, the rates at which various regions of a pasture are grazed — are controlled [1]. The goal is to maximize forage production, or the sustenance and nutrition gained through foraging, which in turn contributes to the maximization of animal production. In order to understand grazing management, one must first understand the players: the animals, and the vegetation.

When left to their own devices at pasture, animals exhibit a tendency toward re-grazing certain areas; some causes of this behavior include the presence of higher-quality vegetation, proximity to water, or the presence of shade on a hot day. Furthermore, vegetation that is regrowing after being grazed is more juicy, palatable, and nutritious than mature vegetation, causing animals to prefer re-grazing areas they have previously visited to grazing areas consisting of more mature growth.

Mature, undergrazed plants have greater stem mass, making them less digestible and nutritious. Overgrazed plants do not possess sufficient leaf mass to generate the energy required for regrowth through photosynthesis; they must obtain this energy from the root structure, resulting in a stoppage of root growth [1].

Thus, forcing a more even distribution of regrazed vegetation is beneficial to the pasture vegetation, as it allows sufficient regrowth time for grazed plants. This is beneficial to the animals, as well, as it helps to maintain even, healthy growth throughout the pasture. However, forcing animals to modify their habitual behavior can cause them stress. A stressed animal will eat and drink less, so introducing stress into the herd can reduce animal production. Therefore, grazing management can be viewed as the mitigation

of the trade-off between maximizing the evenness of pasture coverage while minimizing the stress to cattle.

The most common grazing management strategy worldwide is continuous grazing [2], wherein the cattle are allowed free access to the entire pasture (i.e., cattle stress is totally minimized at the cost of coverage). Continuous grazing is simple and requires minimal overhead. It also provides animals access to plant regrowth in the stages of highest nutritional value. Continuous grazing generally leads to poor grazing distribution (due to the aforementioned animal behaviors). Placement of distributional aids, such as artificial feed and water sources, can often help to minimize this problem [3].

An alternative to continuous grazing is management intensive grazing: a strategy that has been studied and used for at least 25 years, and that is generally considered theoretically sound, but is rarely implemented in its strictest form because it often conflicts with practices already in place. This grazing management strategy relies on the subdivision of the pasture into paddocks through the deployment of permanent and temporary fencing [4]. Permanent fences are used to divide the primary paddocks; temporary fences are used to prevent access to an overgrazed section of the paddock, or to force spot grazing of a previously undergrazed area. The grazing manager has at his disposal the selection of active paddocks or subregions and the stocking rate in each paddock or subregion as tools for implementing intensive grazing. Management intensive grazing requires significantly greater overhead, in the form of fence building and maintenance costs and labor costs, as compared to continuous grazing; in the case of beef cattle, it has been shown to yield an increase in pounds of beef produced per acre of over 533% of typical continuous grazing yields [4].

Management intensive grazing requires moving animals between paddocks approximately once every three days; these paddocks should not be larger in size than 5 acres, with stocking rates of approximately 10 animals per paddock [5]. The time and labor commitment required to meet these requirements are substantial in the context of a small farm or ranch (in fact, many farmers tend to bend these guidelines in practice). In the context of an extremely large ranch, they appear to greatly outweigh the benefits in increased animal production. The King Ranch in Texas is a good example: it consists of approximately 825,000 acres (compare to Rhode Island's size of approximately 776,957 acres) and about 60,000 head of cattle [6]. In a strict application of intensive grazing, these cattle would be split among 1,200 active 5-acre paddocks. Moving these groups of cattle to 1,200 different paddocks every 3 days would be an astounding task for the 60 cattlemen currently employed by the ranch to perform. Furthermore, implementing in-

tensive grazing would require the building and maintenance of a great deal of additional fencing. The King Ranch currently contains an estimated 2,000 miles of fence. Depending on the estimator used, the annual maintenance cost of these fences ranges from $422,400 [7] (assuming the least costly fence type) to $6.6 million [8]. However, if management intensive grazing could be implemented with significantly less overhead, it would mostly likely prove profitable at all scales.

Previous work has shown that virtual fences effectively prevent cattle leaving a virtual paddock, and can be used to move them from one location to another [9], [10], [11] (see Section 2.1.2). Virtual fencing presents the potential for large-scale reduction in fencing and labor costs, as the grazing manager would need only to choose the paddock, or paddocks, he wishes to be active, and allow the automated system move the cattle there and keep them in place. From this point, the next logical step is to automate the grazing management decision-making, as human management, even with virtual fences, would still require managing up to hundreds, or even thousands, of small paddocks.

This work presents a preliminary approach to automating the intensive grazing management process; to the author's knowledge, no other algorithmic solution to this problem has been published. The solution presented here relies on the use of an expert system to model the decision-making of the grazing manager: it determines when, and to where, the cattle should be moved; however, rather than determine the appropriate rules in the traditional manner (by consulting a domain expert), rules are learned via an evolutionary algorithm. Grazing management decisions are often very situation-specific, so it would be difficult to glean generality from a domain expert, whereas an evolutionary algorithm will be as general, or specific, as its fitness function allows. The procedure for evolving expert systems used in this work is modified from and inspired by a number of pre-existing methods, which are discussed in the next chapter. The evolved expert systems used in this work are on-line adaptable, so that they may tailor themselves to the grazing situation at hand. A new approach to performing this adaptation, which attempts to address some inadequacies in previous methods with respect to the grazing management problem, is also presented.

Chapter 2

Related Work

2.1 Computerized Animal Control

Multiple approaches to automated animal containment and motion induction have proven promising or successful. These approaches provide the foundation for artificially intelligent grazing management: the ability to move animals from one location to another automatically.

2.1.1 Herding With Robots

Vaughan et al. have been experimenting with a robotic sheepdog for quite some time [12], [13]. Their technique exploits the predictability of flocking behavior in the animals to be herded in a relatively simple feedback loop algorithm. The robot always maneuvers such that the herd centroid will be located between it and a goal location, with the understanding that the herd will flee from the robot toward the goal. This work was shown to be fairly successful in controlled real-world experiments involving ducks.

The robot sheepdog accomplishes successfully the task set to an actual sheepdog (or to humans, who accomplish it in much the same manner): to gather and move a herd from its present location to a desired goal location, often a barn or pen. To extend this approach to large-scale cattle grazing management, however, would require that many, much larger robots be deployed and work in tandem to move the herd to a new paddock. Alternatively, the use of dynamic virtual fences presents a more attractive solution to this large-scale challenge.

2.1.2 Dynamic Virtual Fences

In virtual fencing, a virtual paddock is defined using a positioning system, such as GPS. Animals are confined within the virtual paddock not by a physical barrier, but rather through the application of stimuli design to discourage them from leaving [9], [10], [11]. Common stimuli include auditory hints (threatening predator noises, or the various whoops and calls used by cattlemen when moving cattle, for example) or electric shocks. Some systems use a combination of stimuli, providing an auditory warning before the virtual fence is reached, followed by electric shocks of increasing severity as the cow crosses the fence. Results have been better with the use of electric shocks, as cattle become aware that the auditory signals are not actually coming from predators or cattlemen.

Dynamic virtual fences are virtual fences that move over time (i.e., the location at which the fence stimuli are applied is changed). Current progress in dynamic virtual fences includes successfully restricting cattle movement within the virtual paddock, moving cattle by slowly moving the virtual paddock, and inter-pose, path-, and fence-planning. Butler et al. successfully influenced the motion of a herd in simulation, and obtained encouraging results with real cattle [9]. They used an exclusively auditory stimulus composed of a library of predator sounds, which were chosen at random. This approach was initially successful, but suffered from habituation.

Anderson et al. used a hybrid audio/electric shock system that did not suffer from habituation [10]. In their experiment, 2 cows were confined with a virtual paddock bounded on three sides by virtual fences, and by a conventional barbed-wire fence on the fourth. The north and south virtual fences moved in tandem at a speed of 1.1m/hr for 10 hours each day. Over the course of the nine-day trial, neither cow successfully escaped the virtual paddock.

Butler et al. added further automation to their virtual fences through an algorithm for planning the corridor between two desired virtual paddock poses [14]. This algorithm uses an A^* search to generate a baseline corridor, which is then modified to avoid obstacles when necessary, or to enclose them if they are small enough. Cattle are guided through the corridor by a series of slow-moving virtual fences; for each leg of the corridor, a new rear fence is created to provide the motion guidance for that leg.

2.2 Evolutionary Algorithms and Expert Systems

A large body of prior work exists in the fields of both evolutionary expert systems and adaptive expert systems. Many methods of evolving expert systems have been developed, operating with various levels of complexity. A number of these methods are discussed here; their drawbacks are discussed in Chapter 3.

2.2.1 Evolutionary Algorithms

An evolutionary algorithm attempts to simulate the biological processes of adaptation, mutation, and natural selection to evolve a solution to a problem. The algorithm consists of a population of individuals, with each individual representing a solution. Individuals are evaluated by a fitness function, and then the fittest individuals are chosen to create the next generation. This process is repeated until the population converges.

The evolutionary algorithm consists of the following components:

Genotype The genotype is the representation of the solution on which the genetic operators act. This is generally an array or tree.

Crossover Operator Crossover is the act of combining two or more parent individuals to produce the same number of new individuals. The most common crossover operator is 1-point crossover, which is most easily understood in the context of an array-based genotype. A cut point n is chosen; the first child's array contains the first n values of the first parent's array, and the values following the first n values of the second parent's array; the second child's array is composed of the remaining values.

Mutation Operator Mutation is the act of randomly altering some part of an individual's genotype. The two most common forms are replacement (e.g., replacing a random value in array with a new value), and swapping (e.g., swapping two randomly selected array values).

Selection Operators A selection operator defines the method for choosing the individuals that will serve as parents for crossover, or the individuals that will comprise the next generation. The selection operator is generally fitness dependent. The simplest selection operator is truncation, i.e., choosing the best n individuals. The most common operator is fitness proportional selection, wherein individuals are selected

randomly with a probability proportional to their fitness. Fitness proportional methods are preferred because they occasionally select unfit individuals, whose genetic material should remain in the global genetic pool to prevent early convergence.

2.2.2 Expert Systems

Here is presented a very brief review of rule-based expert systems. In a rule-based expert system, a set of rules is used, along with the current environment state, to perform inferencing and generate an output or a decision. Generally, each rule is of the form: if x, then y; in a rule of this form, x is referred to as the *antecedent*, and y as the *consequent*. The results of each rule are aggregated in some manner to produce a single output, or multiple outputs or actions may be generated.

Fuzzy expert systems make use of fuzzy sets in order to express rules at a higher level. A fuzzy set is a set whose elements are members of it to certain degrees. This may be formalized by stating that the membership of a variable x in a fuzzy set A is:

$$\mu_{\mathrm{A}}(x) = f(x),$$

where f is a function returning a value in the range $[0, 1]$.

Rules in a fuzzy expert system are generally of the form: if x is in A, then y is in B, where A and B are fuzzy sets with pre-defined membership functions. The degree to which y is in B is determined, through a process known as defuzzification, using the degree to which x is in A. In addition to generating the set of rules, the developer must also specify the fuzzy membership functions. The advantage of this approach over simple expert systems is that the rules read more like a person's reasoning. For instance, consider the following two rules:

1. If it is cold, then I should dress warmly.

2. If it is less than 40°F, then I should cover 90% of my body.

Rule #1 is fuzzy: the words "cold" and "warmly" do not have a specific meaning; however, for the purposes of understanding why the agent utilizing this system has put on a jacket, it is far easier to follow the logic in rule #1 than that in rule #2.

2.2.3 Optimizing Expert System Parameters

Perneel et al. propose a manner for performing expert system optimization on a tiered system [15]. In their expert systems, the rules are grouped into a number of levels, with the rules in each level being related. Furthermore, each rule, and each level, is assigned a weight, which affects that rule's or level's effect on the overall decision-making. Optimization is performed over the overall weighting of the levels, the weighting of the rules within the levels, and the parameters defining the fuzzy membership functions. The chromosome, Θ, is defined as

$$\Theta = \left(\beta_1, \ldots, \beta_n, w_{11}, \ldots, w_{nM_n}, \Theta_{H_{11}}, \ldots, \Theta_{H_{nM_n}}\right),$$

where β_i represents the weight associated with the i^{th} level, w_{ij} represents the weight of rule j under level i, $\Theta_{H_{ij}}$ is a vector containing the membership function parameters for rule j under level i, n is the number of levels, and M_i is the number of rules under level i. Basically, the genetic algorithm is given control over the influence of each rule, the influence of each group of rules, and the shape of the fuzzy membership functions. Thus, given an imperfect expert system generated by a human, the genetic algorithm fine-tunes it to better solve the given problem.

An important component of Perneel's and his colleague's genetic algorithm is the calculation of fitness: how does one determine if a given rule weighting is good? In this case, a *learning database* is used. The learning database consists of a set of decision problems paired with desired decisions. Each expert system's fitness is measured as a sum of absolute error between the system's output and the desired output for each database problem.

Perneel et al. have applied their algorithm to an object recognition problem: classifying targets given two-dimensional infrared images. The expert system must first determine the orientation of the vehicle in the image, and then make the classification.

A basis for results comparison was obtained by building an expert system manually to solve the problem. After optimization, the new expert system outperformed the manual system in 40.5% of cases. In 55.0% of cases, the optimized system performed identically to the original, and in 4.5% of cases, the optimized system performed worse.

2.2.4 Building Expert Systems

A logical extension to the optimization of an expert system through weighting and trimming rules, and through adjusting membership functions, is to

give the genetic algorithm more power. This can mean allowing it to build the rule set itself, to choose the membership functions along with their parameters, or both. Although the genetic algorithm's increased control may allow it to find even better solutions, there is an explosion in terms of the size of the search space when the full contents of the rules and memberships are part of the chromosome. Any approach to evolving an expert system must impose some limitations on the search space, or convergence may never occur.

Evolving New Rules

Drabarek et al. overcome the complexity of evolving a full fuzzy expert system by eliminating the fuzziness [16]. Stated more formally, they utilize threshold membership functions:

$$\mu_{\mathrm{A}}(x) = \begin{cases} 0 & \text{if } x \leq \theta \\ 1 & \text{otherwise} \end{cases},$$

where θ is the threshold constant. Furthermore, the thresholds used are defined a priori, and are not evolved. Thus, the genetic algorithm must only search the space of possible rules; still a vast space, but tractable.

The goal in Drabarek's and his colleagues' work was to evolve new rules for an expert system capable of performing diagnostics on a radar transmitter. Their chromosome consists of two logical parts: the rule antecedent (the "if" clause), and the rule consequent (the "then" clause). Each antecedent is a vector of integers in the range $[-1, 1]$, where each integer represents one of 10 important symptoms (these are the connections between circuit components). A value of 1 indicates that the symptom should be normal, a value of 0 indicates that the symptom is not considered, and a value of -1 indicates that the symptom should be abnormal. Each consequent is a vector of integers in the range $[0, 1]$, where each integer represents a symptom or circuit component that should be checked. A value of 1 indicates that the component or symptom should be checked, and a value of 0 indicates that it should not be checked. The whole chromosome is the concatenation of the antecedent part and the consequent part.

As in the work of Perneel et al., the calculation of fitness is both important and non-trivial. Rather than providing training data, Drabarek et al. use a set of heuristics to determine if a given rule is acceptable. Rules are judged by the following criteria[1]:

[1]Paraphrased from Drabarek et al. [16].

1. A path from each incorrect antecedent symptom to each consequent component should exist in the circuit. Shorter paths are better.

2. A backward path from each consequent object to each correct antecedent symptom should exist in the circuit.

3. Rules in which checking a consequent component is recommended, despite all symptoms leaving that component being correct, are penalized.

These heuristics reward rules that appropriately relate symptoms and components, and that do not recommend checking clearly operational components, which is wasteful.

Testing experiments were run, in which the system was provided with information about two symptoms connected to a single component (one is an input, the other is the output). The component has two other inputs. After running the genetic algorithm, rules indicating that the component should be checked if any combination of its input symptoms are correct and its output symptom is incorrect were evolved. The number of generations required to obtain these rules, and the completeness of the generated rule set, were both decreasing functions of population size. The system may be considered a success, as it generated logical diagnostic rules; however, the problem of rule generation was applied to a problem where the full, optimal rule set could easily be generated manually (and, in fact, the authors provide this rule set).

Separate Evolution of Rules and Membership Functions

The previous section's result indicates that rules can be evolved, but without considering membership functions. In this section, two approaches to evolving both rules and memberships are discussed. As previously stated, the problem of an explosion in search space complexity arises. Both methods presented here account for this issue by evolving the rules and membership functions separately, admitting an additive increase in complexity, rather than a multiplicative one.

Amaral et al. use a serial approach: first, rules are evolved; then, memberships are evolved [17]. When evolving the rule base, the chromosome consists of $N + 5$ values: the consequent values for each of the N possible rules, and 5 additional parameters. The additional parameters define the implication and aggregation operators, the defuzzification method, and the t-norm and t-conorm (these define how conjunction and disjunction work

in fuzzy logic). All possible rules are dealt with in each chromosome because the solution is being applied in evolutionary hardware. Thus, each rule actually represents a possible input to some configurable hardware, and the consequents are the hardware's outputs. Fitness for the first round is determined using "traditional metrics" (mean squared error or root mean squared error); however, the paper does not discuss how expected values are determined: the process is most likely application-specific.

Once the rules have been evolved, a separate genetic algorithm is used to optimize the fuzzy membership functions. In this step, the chromosome consists of a vector of values identifying the vertexes of triangular membership functions (3 vertexes per function). Once again, the details of fitness evaluation are rather vague.

Amaral et al. tested their methodology by applying it to a simulated evolutionary hardware problem. Two experiments were conducted, wherein the evolutionary fuzzy system was asked to model two systems. For comparison, the experiments were also performed on a neural-fuzzy system, called NEFCON, and a neural-genetic-fuzzy system, called NEFCON-GA. NEFCON-GA had the best performance; the authors point out, however, that the NEFCON-GA produced rather irregular fuzzy membership functions, which make the system more difficult for a human to interpret. The evolutionary fuzzy system's more interpretable memberships demonstrate the trade-off between performance and interpret-ability.

An alternative to serial evolution of rules and memberships is parallel evolution (co-evolution). Akbarzadeh-T et al. propose a co-evolutionary approach to evolving fuzzy systems, wherein the rules are evolved via genetic programming, and the memberships are evolved by a genetic algorithm [18]. Here, search space complexity is lessened through the use of a binding matrix, which determines which rule sets combine with which membership function sets (not all combinations are tried). Furthermore, the concept of "friendship" is introduced: when a specific rule set/membership set pair performs particularly well, its bond is weighted.

The main focus of Akbarzadeh-T's and his colleagues' paper is computing the fitness of combined individuals. Each rule set's fitness is calculated as the weighted average of its performance when coupled with each membership function, where the weights are determined by the level of friendship that exists between each rule set/membership set pair. The performances are also used to determine modifications to these friendship levels. The fitness for each membership set is computed in a similar manner.

The co-evolution method was tested by using it to evolve an expert system capable of modeling the Mackey Glass chaotic time series. The

expert system is given the last four values of the series and must output the next value. The proposed method was largely successful, and outperformed both a static expert system and an expert system consisting of uniform membership functions and rules evolved using genetic programming.

Evolving Rules and Membership Functions Together

Calvo et al. developed a system for obstacle avoidance that uses a constructive neural network with a fuzzy system component [19]. A genetic algorithm builds a fuzzy system to determine when neurons should be added to the network (there is a trade-off between increased performance and increased complexity when adding a neuron); performance of the fuzzy system is based on the number of nodes it adds to the neural network during an obstacle avoidance task. In this genetic algorithm, rules and memberships are evolved at the same time, in the same individual. The search space is limited by the fact that there are only four inputs to the fuzzy system, and only three possible memberships for each input to each rule.

The fuzzy system being evolved is not an expert system, but rather it is a classifier system. In this type of system, only one rule is executed: the one with its antecedent part most closely matching the input. In the specific application explored by Calvo et al., the winning rule's consequent value is then used to determine whether a new neuron should be added to the neural network.

Each individual represents a set of rules and membership function parameters; a single rule is encoded as:

$$[V_0, V_1, V_2, V_3, A_0, A_0, A_0, A_1, A_1, A_1, A_2, A_2, A_2, A_3, A_3, A_3, C],$$

where each V_i indicates the set to which input i should belong ($0 =$ LOW, $1 =$ MED, and $2 =$ HIGH), the A_i values determine the shapes of the membership functions for the fuzzy variables, and C is the rule consequent. Reproduction consists of one-point crossover and mutation.

Calvo et al. present an interesting approach to creating new generations. A new generation is not created until the performance of the neural network decreases. Performance is determined by the number of collisions that occur while attempting to navigate to a target. If performance is not decreasing, then the classifier system is properly adjusting the neural network, and so should not be modified.

The proposed system was tested in multiple simulated environments and compared to the performance of a constructive neural network that adds a neuron each time a collision occurs. The evolved fuzzy system was able

to build a neural network with half as many nodes that could successfully collect targets and avoid obstacles nearly all of the time.

The approach to evolving expert systems developed for this work lies in between those discussed in the last two sections: the rules and memberships are evolved via a single evolutionary algorithm; however, the crossover and mutation operators always handle rules and memberships separately.

2.2.5 A Hybrid Hybrid Approach

In this section, an approach from Shi et al. is presented that combines ideas from the previous one and from Xu and Vukovich [20]: evolving the full expert system, both rules and fuzzy memberships, and using an expert system to optimize the genetic algorithm parameters [21]. The only search space limitation imposed in this system comes from the fixed rule complexity (this has been a feature common to most of the systems presented: rules perform only a simple combination of the inputs).

In the work of Shi et al., the chromosome is represented as a string of integers. These integers encode the number of rules in the system, the fuzzy membership function types (there are six) and parameters for each input, and the rules. A maximum number of allowed rules is calculated a priori, and each chromosome encodes this many rules; the rule count encoded at the beginning of the chromosome indicates how many of the encoded rules are actually used; the rest are ignored, but are still encoded in order that the chromosomes all be of uniform length.

The rules encoded within a chromosome are not necessarily valid, so a form of genetic repair must be performed. In this work, invalid rules are simply removed from the system when it is built from the chromosome data. Invalid rules are defined as those containing either no antecedent or no consequent part. If all rules are removed from an individual in this manner, it is assigned an arbitrary low fitness.

Fitness is computed using training data. An application-appropriate error metric must be determined. In the case of function estimation, the authors recommend a relative error function, so that the magnitude of the target output does not affect the error calculation. For classification systems, the authors recommend a comparison of correct classifications to incorrect ones.

Shi et al. use an adaptive genetic algorithm to decrease convergence time. The genetic algorithm's adaptation is accomplished through a static expert system, which is designed to mitigate the trade-off between performing mutation and crossover. The overall goal of the genetic algorithm parameter

adaptation is to encourage crossover during the early generations, when fitness is low, and to encourage mutation during the later generations, when fitness is high. The motivation behind this is that mutation can tweak an already fit individual, while crossover is best for quickly changing large parts of an individual. Thus crossing fit individuals has a good chance of destroying what makes them fit, while mutating unfit individuals will change them little over many generations.

The system has been applied to a classification problem: classifying iris specimens by species. The authors ran a number of simulations, testing with various restrictions on their system (fixing the memberships, turning off genetic algorithm parameter adaption, and both). The system that evolved both rules and memberships, and used the adaptive genetic algorithm, had the best overall performance both in correctness of classification and in convergence time.

2.2.6 Adaptive Expert Systems

The methods described in the previous sections all modify the expert system through indirect learning, i.e., learning is ultimately accomplished through random modifications that eventually lead to a better result. This section presents two methods for performing learning, or other modifications, in a direct manner: adaptive-network-based fuzzy inference systems (ANFIS), which use backpropagation learning to arrive at optimal rules, and meta-rules, which are rules built into the expert system itself that operate on other rules, rather than on the expert system output.

Adaptive-Network-Based Fuzzy Inference Systems

ANFIS is a methodology proposed by Jang [22] that combines fuzzy inference systems (expert systems) and artificial neural networks, in order that the inference system rules and memberships may be learned from input/output training pairs. Although it is primarily an off-line methodology, the ANFIS may be retrained at regular intervals to allow it to adapt to new information. The ANFIS performs direct learning in the sense that its learning mechanism deliberately reduces the difference between the desired and actual outputs. Thus, the system's desired output must be known for each input vector; however, as is made more clear in Chapter 3, the output of the proposed expert system is not known, and its validity can only be determined indirectly. Therefore, direct modification must be accomplished through other means.

Meta-Rules

Meta-rules provide a mechanism for altering (either permanently or temporarily) an expert system in an on-line manner based upon current input, or other stored or gathered information. Meta-rules are often used to restrict the number of rules that actually fire for any given input, usually to increase the efficiency of the inference engine [23, 24]. They can also be used to add rules to, modify rules in, and delete rules from the system. Jankowska [25] discusses the use of meta-rules for detecting and resolving contradictions between rules, a problem that can arise when rules are added to large, pre-existing knowledge bases.

Chapter 3

Algorithm Design

The automated grazing management solution presented in this chapter is comprised of two major parts: the management algorithm, which actually performs grazing management, employing an adaptive expert system for decision-making, and the evolutionary algorithm, which is used to learn the best expert system given some fitness metric.

The evolutionary algorithm and on-line expert system adaptation algorithm complement one another by performing global and local learning and adjustment, respectively. The evolutionary algorithm operates on a metric supplied at the end of testing, representing the overall performance of the system; the on-line adaptation algorithm operates throughout testing, making adjustments to the system based on observed cause-effect relationships while it runs. Furthermore, because the procedure through which the evolutionary algorithm learns is ultimately random, whereas the on-line adaptation is based on reasoning, together the two algorithms account for two forms of solution modification occurring during human problem-solving: identifying that the current solution is inadequate and experimenting with possible changes to see if it can be improved (evolutionary algorithm), and deducing logically corrections to be made to the current solution based upon observations of its effectiveness (adaptation algorithm).

3.1 Herding Algorithm

A herding algorithm using an expert system to perform decision making has been developed. The algorithm adapts itself in an on-line manner by drawing conclusions from observed cause-effect relationships, in order that a general expert system may be applied to more than one specific herding situation.

MANAGE-GRAZING

1 **while** true
2 **do for** each cell $C_i \in \{C_{\text{current}} \cup \text{NEIGHBORS}(C_{\text{current}})\}$
3 **do** $R[i] \leftarrow \text{SCORE}(C_i)$
4 GOTO $\left(C_{\text{argmax}_i\{R[i]\}}\right)$
5 WAIT(t)

Figure 3.1: High-level grazing management procedure. t is the time to wait between iterations.

Herd behavior shares a good deal of commonality; however, differences in climate, time of day, and cattle breed, among others, can affect the observed herd behavior. On-line adaptation allows a general algorithm to be deployed in these various situations and tweak itself based upon runtime observations.

3.1.1 General Procedure

The procedure described in this section is inspired by the indirect coverage algorithm described by Pirzadeh and Snyder [26]. In indirect coverage, the environment is split into a grid. The number of visits to each cell is stored; when deciding which cell to visit from the current one, the cell with the fewest total visits is chosen. Eventually, the whole environment will be visited at least once. The herding algorithm splits the field into a grid of large cells (each cell should be large enough to allow the cattle to graze for awhile before they must be forced to move; management-intensive grazing dictates 1–5 acres per paddock), and keeps the cattle within one cell using four virtual fences deployed along the edges of the current cell, each having infinite length. At a well-defined interval, an expert system is used to calculate a score for the current cell and each cell within the current one's 4-neighborhood. If the current cell's score is highest, then nothing is done; otherwise, the cattle are moved slowly to the cell with the highest score. This process is repeated indefinitely, and is defined formally in Figure 3.1.

The GOTO function is quite simple. Since all virtual fences are infinite in length, and since movement is always in a cardinal direction, only two fences have to be moved. The fence adjoining the new cell is immediately moved to the far end of that cell, so that the cattle may begin moving into the new cell unhindered. The fence at the opposite end of the current cell is slowly moved to the other edge of the new cell at a speed of 0.025m/s; this

speed is slow enough that if any cattle are shocked by the moving fence, they will be able to move away from it quickly enough to avoid being repeatedly shocked. The herding algorithm is paused until moving to the new cell has completed.

SCORE (C_i) is calculated using an adaptive expert system obtained from an evolutionary algorithm. Each rule in the expert system is of the form: "if *expression*, then y," where y is a real value. Let R^* define the set of all rules, and the function $y(R)$, for some $R \in R^*$, denote the output value obtained from multiplying the y value in R's consequent by the overall fuzzy value of R's antecedent. Then, the total score for the cell C_i is SCORE $(C_i) = \sum_{R \in R^*} y(R)$. Additionally, the current cell's score includes a constant bonus representing the inherent reduction in cattle stress obtained through not moving them.

3.1.2 Variables

This section describes the variables available to the adaptive expert system. The following variables are cell-specific:

VEGETATION The mean vegetation value for the cell.

WATER The percentage of the cell consisting of water.

OBSTACLES The percentage of the cell that is impassable.

EVENNESS The difference in elevation between the highest and lowest points in the cell, expressed as a percentage of the maximum possible difference.

HAPPINESS Perceived happiness level of the cattle in the cell; this is calculated as $1 - \min\left(1, \frac{z}{z_{\max}}\right)$, where z is the number of shocks recorded in the cell for the given time period, and $z_{\max}$ is the number of shocks per time period considered to represent massive stress levels among the cattle (the value used is 1 shock per second).

CELL-AGE A value representing both how long this cell has been the current cell, and how long it has been since this cell was the current cell.

The following variables are equivalent across all cells:

LINEARITY A value representing how linear the path taken by the algorithm has been; this is calculated as the ratio of the number of cells along one axis that have been visited since the last axis change to the total number of cells along the longest axis of the field.

HERD-DISPLACEMENT The ratio of the distance the herd was displaced during this time interval to the maximum displacement possible within the cell and time interval. A very small herd displacement indicates that the cattle are bunching at a fence that they wish to cross; conversely, a very high displacement indicates that the cattle are moving farther or faster than they normally would to find food or water.

The HAPPINESS and CELL-AGE variables for each cell are modified over time. CELL-AGE is decremented by a constant value for the current cell and incremented by a constant value for all other cells (it is capped within the range $[0, 1]$). The decrement for CELL-AGE is larger than the increment. HAPPINESS is set for the current cell using the aforementioned calculation. For each cell, HAPPINESS is incremented by a constant value (once again, it must remain within the range $[0, 1]$). These increments and decrements occur each time the overall algorithm loops.

3.1.3 Adaptive Expert System

The adaptive expert system consists of three parts: the rule set, the fuzzy membership functions, and threshold value θ. The rules are used to calculate a score for a cell, as described above. Fuzzy membership functions are used to determine the extent to which the cell's raw data belong to the variables described in the previous section. The θ value is used in the adaptive algorithm.

Rules

The expert system's rules are all of the form "if *expression* then y," where y is the amount to add (or subtract) from the total score when *expression* evaluates to 1. The *expression* is a fuzzy expression composed of operators and operands. Operands take the form of a fuzzy membership query for a specific variable. Table 3.1 describes the possible operators.

Membership Functions

Three trapezoidal membership functions are stored for each variable, defining that variable's fuzzy membership in the sets LOW, MED, and HI. The only limitations imposed upon the definition of these functions are that the LOW and HI sets may not overlap, and that for an input x, $\mu_{\text{LOW}}(x) + \mu_{\text{MED}}(x) + \mu_{\text{HI}}(x) = 1$.

Operator	Syntax	Evaluation
and	*expr1* and *expr2*	$\min(expr1, expr2)$
or	*expr1* or *expr2*	$\max(expr1, expr2)$
not	not *expr*	$1 - expr$
current-cell	current-cell	1 if the cell being processed is the current cell, 0 otherwise.
reachable	reachable	1 if the cell being processed is directly reachable from the current cell, 0 otherwise.

Table 3.1: Operators found in expert system rules.

On-line Adaptation

The adaptive algorithm maintains a directed graph associating the current cell's environment with the current perceived happiness of the cattle; thus, this graph provides a rudimentary representation of cause and effect. The current cell state at time t is represented by the vector

$$S_C^t = [V, O, W, A, E]$$

where V, O, W, A, and E are integers denoting the fuzzy set (LOW = 0, MED = 1, HI = 2) to which the variables VEGETATION, OBSTACLES, WATER, CELL-AGE, and EVENNESS, respectively, *most* belong. Likewise, S_H^t is the set to which HAPPINESS most belongs. The following multi-sets hold the cumulative state:

$$S_C^* = \bigcup_t S_C^t, S_H^* = \bigcup_t S_H^t$$

The adaptive graph $G_{S_C^* \to S_H^*}$ is a weighted, directed graph mapping cell states to perceived happiness with weights indicating the number of occurrences of a specific mapping. Thus,

$$V\left(G_{S_C^* \to S_H^*}\right) \subseteq S_C^* \cup S_H^*$$

$$E\left(G_{S_C^* \to S_H^*}\right) = \left\{(u, v, w) \,|u = S_C^t, v = S_H^t, w = n_{(u,v)}\left(\bigcup_t (S_C^t, S_H^t)\right)\right\}$$

where the function $n_a(B)$ = the number of occurrences of a in multi-set B.

The algorithm to populate $G_{S_C^* \to S_H^*}$, determine when learning takes place, and update the rule set is described in pseudocode in Figure 3.2. Conceptually, the adaptation algorithm identifies cause-effect relationships

ADAPT

1 Obtain S_C^t and S_H^t
2 **if** $(S_C^t, S_H^t, w) \in E\left(G_{S_C^* \to S_H^*}\right)$ for some w
3 **then** $w \leftarrow w + 1$
4 **else** $E\left(G_{S_C^* \to S_H^*}\right) \leftarrow E\left(G_{S_C^* \to S_H^*}\right) \cup \{(S_C^t, S_H^t, 1)\}$
5 $(S_C, S_H, w) \leftarrow \max_{(u,v,w) \in E\left(G_{S_C^* \to S_H^*}\right)} \{w\}$
6 **if** $w > \theta$
7 **then for** each rule R in the expert system
8 **do** $V \leftarrow$ the set of variables referenced in R
9 $d \leftarrow ||V - S_C||$
10 **if** $S_H = 2$
11 **then** $\Delta y \leftarrow d \min\left\{1, e^{-\frac{R_y+4}{4}}\right\}$
12 **if** $S_H = 0$
13 **then** $\Delta y \leftarrow -d \max\left\{1, e^{-\frac{4-R_y}{4}}\right\}$
14 $R_y \leftarrow R_y + \Delta y$
15 $E\left(G_{S_C^* \to S_H^*}\right) \leftarrow E\left(G_{S_C^* \to S_H^*}\right) - \{(S_C, S_H, w)\}$

Figure 3.2: Pseudocode describing the expert system adaptation algorithm.

of significance (the system's θ value is the threshold for determining significance; see lines 1–6), and then modifies the consequent values of the rules based upon their relevance to the cause-effect relationship identified, and whether the effect (perceived cattle happiness) is positive or negative (lines 8–15). The relevance of a rule to a cause-effect relationship is measured by determining the similarity between the cause (environment state) and the state the rule attempts to match.

In the pseudocode, R_y is used to represent the consequent value of the rule R. The variable set, V, for rule R is composed of ordered pairs (v, s), where v is one of the system variables, and s is one of the three fuzzy sets. For each operand in R of the form $\mu_s(v)$,

$$V = V \cup \begin{cases} \bigcup_{z \in \{0,1,2\}-s} \{(v, z)\} & \text{if } \mu_s(v) \text{ is within a } \textit{not} \text{ operator,} \\ \{(v, s)\} & \text{otherwise.} \end{cases}$$

The similarity, d, between S_C and V is computed as

$$d = \sum_{(v,s) \in S_C} D((v,s), V)$$

$$D((v,s), V) = \begin{cases} 1 & \text{if } (v,s) \in V, \\ 0 & \text{otherwise.} \end{cases}$$

Note that d measures *only* the similarity between the two sets, and not the dissimilarity (i.e., the presence of additional rules in V does not detract from the similarity score).

The exponential functions in the consequent modification equations (lines 11 and 13) are designed to change the consequent value more drastically when its value is significantly different from the ideal. In the case that a positive relationship has been identified (S_H is 2, or HI; line 10), rules matching the environment that produced HI HAPPINESS should increase the cell's score. Thus, rules currently having a negative consequent value will see a large increase, whereas rules already possessing a positive consequent will see a smaller one. This behavior is intended to mimic a human modifying his solution to a problem: in the case that the solution is "way off," he may make large changes; however, if the solution is close, but not quite right, he will mostly likely only tweak it slightly. In the case that a negative relationship has been identified (line 12), the same method is used, but the ideal consequent value is a negative one.

3.2 Evolutionary Algorithm

The method for evolving expert systems described here performs a more powerful expert system refinement than that presented by Perneel et al. [15]: a set of seed systems is used in generating the initial population to help limit the search space to viable systems, but the evolutionary algorithm is able to add, modify, and remove rules, as well as being able to alter membership functions. Furthermore, unlike the methods described in the previous chapter, the evolutionary algorithm operates on complex rules potentially containing nested operators. These extensions allow the evolutionary algorithm to explore a much larger portion of the total potential solution space than the aforementioned approaches; as the algorithm is given more freedom to explore this space, it becomes less necessary that the set of systems used to seed the initial population contain a wide variety of genetic material, thus allowing the evolutionary algorithm to shoulder a greater portion of the problem-solving responsibility.

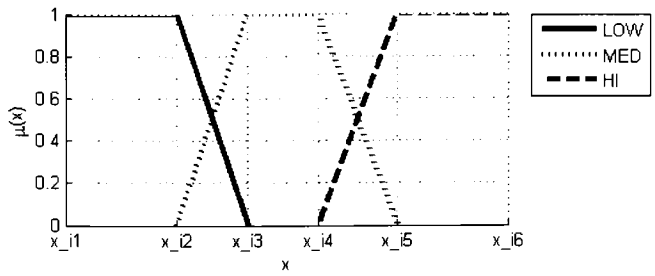


Figure 3.3: Mapping of the six membership function parameters to three trapezoidal fuzzy membership functions.

3.2.1 Genotype

The phenotype for the evolutionary algorithm is the expert system itself, which was described in the last section. The genotype contains the same information; however, in order to discuss genetic operators, a description of the genotype's data representation is required.

Each rule is composed of two parts: the antecedent and the consequent. The antecedent is an expression containing operators and operands; since an operand may be an expression in and of itself, the antecedent is represented by a tree, as is generally done in genetic programming. The consequent is a single value, and so is stored as such. The genotype maintains a set of pairings of antecedent trees and consequent values to represent the rules set.

For each variable defined in Section 3.1.2, three fuzzy memberships must be defined, for the fuzzy sets LOW, MED, and HI. The parameters defining each function are all stored in a single array of the form:

$$[x_{11}, x_{12}, x_{13}, x_{14}, x_{15}, x_{16}, x_{21}, \ldots, x_{n6}]$$

The values x_{i1} through x_{i6} define the three trapezoidal membership functions for the i^{th} variable; this is shown in Figure 3.3.

3.2.2 Crossover

Crossover is performed separately on rule sets and membership functions; when two genotypes undergo crossover, both their rules and memberships are affected. For the rule sets, 1-point crossover is used at the set level (i.e., the first n rules are kept, and the remainder are swapped with the other parent). Swapping of parts of rules occurs during mutation.

1-point crossover is also used for membership functions; however, the only allowable cut-points are those occurring between individual functions.

This limitation prevents the need for post-crossover genetic repair, because, within a specific function definition, the function parameters must be non-decreasing.

Finally, the adaptive θ threshold undergoes 1-point crossover at the bit level.

3.2.3 Mutation

Mutation is performed by choosing one of the following operations at random:

SWAP-PARTS Swaps subtrees between two randomly selected rules.

MUTATE-MEMBERSHIPS Chooses a membership function at random and replaces it with a new, randomly generated membership function. The function generation algorithm ensures that the new parameters are non-decreasing.

ADD-RULE Generates a new rule. Starting with the root node, each node has a 75% chance of becoming an operand. Once all of the tree's leaves are operands or operators taking no arguments, the rule is complete. (Note: the algorithm is prevented from generating rules with only the "current-cell" operation).

REPLACE-RULE Removes a random rule and generates a new rule to add in its place.

REMOVE-RULE Removes a random rule.

MUTATE-CONS Chooses a new consequent value for a randomly chosen rule. The new value is chosen randomly from a uniform distribution with range $[-4, 4]$.

MUTATE-THETA One bit in the binary representation of θ is flipped.

3.2.4 Parameters

Table 3.2 defines the parameters used by the evolutionary algorithm. Fitness proportional selection is often preferable to truncation selection due to the chance of selecting unfit individuals. This is because an individual may be globally unfit, but still possess useful genetic material; excluding these individuals can result in premature convergence due to a lack of diversity

Parameter	Value
Population Size	16
Mutation Rate	0.05
Crossover Rate	0.75
Elitism Rate	0.375
Parent Selection	Fitness Proportional
Elitist Selection	Truncation
Survivor Selection	Fitness Proportional

Table 3.2: Parameters used by the evolutionary algorithm.

among genetic material. However, truncation selection was chosen for picking the individuals to carry over from the previous generation because this method guarantees that the best individuals are chosen. The full spectrum of genetic material from the previous generation still has the possibility of being represented through crossover, as unfit individuals may be selected as parents.

3.2.5 Fitness Calculation

Three fitness metrics were designed and used in experimentation: coverage-based, stress-based, and combined. The separate coverage-based and stress-based metrics were used to determine if different rule sets would be generated given these different goals. The combined metric forces the evolutionary algorithm to compromise between maximizing coverage and minimizing stress.

Coverage-Based Fitness

Coverage-based fitness requires that ideal coverage be defined. Here it is defined in terms of the distribution of vegetation levels across the field at the end of simulation; in order to determine the ideal distribution, the relationship between the cattle and the forage matter, and the needs of each, must be considered.

Cattle obtain the greatest nutritional value from re-grazed vegetation, because when a plant is growing, it is more digestible and provides more protein; also, plants in earlier stages of growth tend to be juicier and taste better to the cattle. Mature plants have greater stem mass, which makes them less digestible, and contain more fiber, making them less palatable [1], [2]. Turner et al. [4] indicate that undergrazing has occurred when plants

are allowed to grow to larger than approximately 75% of their maximum size.

When too great a percentage of a plant's leaf volume is removed through grazing, however, it must consume nutrients stored in its roots to regrow leaves. This results in root growth stoppage, which in turn can lead to a weakened root system, loss of plant species, and a reduction in forage yield for the animals. Turner et al. and Poole [1] both indicate that overgrazing occurs when plants are allowed to be reduced to less than approximately half their maximum size.

Therefore, the desired vegetation distribution may be modeled as a normal distribution with a mean value of 66% of the maximum vegetation value. This means that the average plant in the field is not so early in its growth cycle that it must suffer root growth stoppage to continue leaf growth if grazed, and it is also not so mature that it suffers a reduction in nutrition or palatibility. Thus, coverage-based fitness is calculated as the raw χ^2 test score when comparing the distribution of the final vegetation levels of the field to the ideal distribution, defined as a normal distribution with $\mu = 168.3$ and $\sigma = 49.7$ (where the maximum vegetation level is 255). A lower χ^2 score indicates a better match between the data and the desired distribution, so the evolutionary algorithm should attempt to minimize this fitness metric.

At the completion of a simulation run, the vegetation map is split into 20 meter by 20 meter cells. Cells with fewer than 50 non-zero values are ignored, in order to remove major obstacles from consideration. An average level is taken for each cell, and the averages are collected in a histogram. Finally, this histogram is used to perform a χ^2 test, whose result represents the fitness of the expert system in question.

Stress-Based Fitness

Stress-based fitness is calculated similarly to HAPPINESS, except that here it is *unhappiness* that is being determined. For each 15-hour segment of runtime, the number of shocks per second is calculated (capped at 1); overall unhappiness is the mean of these values. The goal of the evolutionary algorithm is to minimize this metric, as well.

Combined Fitness

The combined fitness metric calculation is:

$$f = \chi^2 e^u$$

where χ^2 is the coverage-based fitness and u is the stress-based fitness. u is taken exponentially to place greater emphasis on low unhappiness values (i.e., to punish severely a system whose overall unhappiness is high). Once again, the evolutionary algorithm's goal is to minimize this metric.

Fitness Short-Circuiting

In order to save on both time and processing, a short-circuiting mechanism is introduced to the fitness calculation. If no attempt to move the cattle is made within 30 hours (this timeframe chosen for the specific simulation environment), then the system in question is arbitrarily assigned fitnesses of 50.0 and 1.0 for the coverage- and stress-based metrics, respectively, yielding a combined fitness score of 135.914 (compare this to a typical set of metrics for a very poor system: $\chi^2 \approx 22.0$, $u \approx 0.5$, $f \approx 36.272$).

3.2.6 Initial Population

A small set of human-produced individuals is provided to the initial population algorithm, allowing a human to introduce rules that he believes will be useful, or to introduce systems to be refined. Two systems are selected randomly from the set of seeds and are crossed over. Each of the resulting children is mutated. The mutated children are added to the set of seeds, and the process is repeated until the cardinality of the seed set is twice the desired population size. At this point, individuals are selected at random until the desired population size is reached; these become the initial population; the remaining seeds are discarded.

Chapter 4

Implementation and Evaluation

4.1 Simulator

A simple cattle simulator was developed for testing. This section describes its final functionality; during testing, functionality and complexity were introduced into the simulator incrementally in order to determine system performance given various sets of variables. Later sections detailing the various experiments performed describe what simulator functionality was present.

The overall algorithm governing the simulated cattle is quite simple: a number of forces are generated representing attraction and repulsion from various stimuli. These forces are added, adjusted for maximum turning rate and speed, and applied to the cow. This process occurs for each second in simulation time.

Much of the simulator functionality and implementation was provided by Dr. Butler, as the simulator's development is outside the scope of this work. Specifically, the algorithms described in the sections concerning intra-herd forces, vegetation-related forces, obstacle avoidance and fence stimuli reactions, turn and speed correction, and vegetation consumption and regrowth are Dr. Butler's work. They are described here in order to give the reader a better understanding of the experiments conducted.

4.1.1 Forces

Intra-Herd Forces

In order to maintain the herd structure, the cattle experience repulsion between one another and attraction to the herd centroid. For cows C_1 and C_2 having positions P_{C_1} and P_{C_2}, the repulsive force on C_1, $f_r^{C_1}$, is calculated as follows:

$$r = K_r \frac{(P_{C_2} - P_{C_1})}{||P_{C_2} - P_{C_1}||^2}$$
$$f_r^{C_1} = -(r \cdot \max(0, (P_{C_2} - P_{C_1}) \cdot v_{C_1}))$$

where K_r is the repulsive constant, r is the intermediate force, and v_{C_1} is cow C_1's velocity.

The attractive force on C_1 toward the herd centroid is calculated as

$$f_a^{C_1} = \begin{cases} K_a (P_H - P_{C_1}) (||P_H - P_{C_1}|| - d) & \text{if } ||P_H - P_{C_1}|| > d, \\ 0 & \text{otherwise} \end{cases}$$

where K_a is the attractive constant, P_H is the herd centroid, and d is the minimum distance from the herd centroid at which the cow is attracted to the herd.

Herd Drift

The simulated cattle are subject to a drift force, which is applied uniformly across the herd. This force is designed to replicate two behaviors seen in grazing cattle: first, the tendency of leader and rogue cattle to cause the herd to drift; second, the periodic migration of cattle between various favored locations. The simulator achieves both behaviors by applying to each cow a weak force from the herd centroid to a favored location. Each favored location is assigned a probability of being chosen, p_i, which is used in conjunction with a time interval in seconds, I, to determine which favored location to use and for how long. If the i^{th} location is chosen, then it is the target of the drift force for a period of $\lfloor p_i I \rfloor$ seconds, after which a new favored location is chosen.

Vegetation

The simulated cattle are not specifically attracted to vegetation, as this would involve computing attractive forces toward a vast majority of the

field; however, vegetation levels (and qualities) in front of the cow, and to the left and right, are considered to determine if the cow should continue on its current course, or turn, to find food. First, values very near to the cow are checked: directly ahead, and 45° to the left and to the right. If one of these three values is great enough, then that direction is taken; otherwise, far off values are considered. These values are checked at the same three angles, but at a distance of 5 times the cow's current velocity. Searching in the second scope allows the cattle to escape over-grazed patches.

The vegetation values considered in making the decisions described above take into account both the amount of vegetation present and its quality. Vegetation quality is represented in simulation by the assignment of different multipliers to different areas of the field. The final vegetation value is the amount of vegetation times the multiplier; thus, a larger multiplier increases the apparent amount of vegetation, representing the greater availability of nutrients in higher-quality vegetation.

Elevation

Cattle, like most creatures, will prefer to travel by a level path if given the choice. Therefore, the simulator maintains a height map for the field, and each cow experiences repulsive forces proportional to the difference in elevation between its location and those nearby.

Water

An attractive force, f_w, from each cow to the nearest water in the field is calculated, with magnitude determined as follows.

$$||f_w|| = K\left(1 - \frac{\Delta t}{I_w}\right)^2$$

where K is a constant intended to keep this force's magnitude within the same order of magnitude as the other forces', Δt is the elapsed time since the cow last drank, and I_w is a constant defining the expected elapsed time between periods of drinking. Thus, as more time elapses since the cow last drank, its desire to seek water increases quadratically, and, once I_w is exceeded, the attractive force toward water quickly becomes dominant.

Cattle drink, on average, around once every 6 hours. In the simulator, since the cattle always eat, time is considered to be divided by 3 (i.e., the time spent sleeping, resting, or wandering without grazing is ignored); thus, I_w is typically 2 hours.

Obstacles and Fences

In the simulator, obstacles include pre-existing real fences, trees, rocks, impassible water, etc. Obstacle avoidance in the simulator is quite simple: if an obstacle is present in front of a cow at a very close distance, a very large force is applied to cow in the opposite direction.

Stimulation in response to crossing a virtual fence is applied as though the cow were wearing a collar designed to apply an electric shock to one shoulder or the other; this provides the cow with some information regarding the direction it should turn to most quickly return to the correct side of the fence. In simulation, this is achieved by looking at the cow's heading and the orientation of the crossed fence to determine which side of the cow is closer to the correct side of the fence, and applying to the cow a force perpendicular to the cow's heading in that direction. Specifically,

$$\alpha = h \times n$$

$$f_{\text{shock}} = \begin{cases} \langle -h_y, h_x \rangle & \text{if } \alpha < 0, \\ \langle h_y, -h_x \rangle & \text{otherwise.} \end{cases}$$

where h is the cow's heading and n is the fence's normal (a vector perpendicular to the fence and in the direction of the forbidden side).

Turn and Speed Correction

Once all of the forces to be applied to a cow have been calculated and added together, the simulator must determine if applying this overall force to the cow would result in too drastic a turn or too great a speed. First, the turning case is addressed. The cow's potential torque is calculated as $\tau = \frac{F}{\|F\|} \cdot h$, where F is the total force being applied to the cow, and h is its current heading. If F is large enough (i.e. the cow would be moving with some velocity, and not just turning in place), and the value of τ indicates a turn of greater than 45°, then a turn of 120° at 0.5m/s is initiated; in effect, the cow slows to turn, so that on the next iteration, the combined forces will cause it to move forward along the desired heading.

If the turn indicated by τ is not too great, then the overall force is combined with the cow's current velocity, thus introducing momentum. This new force is capped at a pre-determined maximum velocity of 2m/s if necessary. If the magnitude of the new overall force is not greater than the maximum velocity, but is greater than the typical walking speed (defined as 0.1m/s), and the cow was not shocked during this iteration, then the new

overall force, F_n, is adjusted in the following manner:

$$F_n = F_n \left(1 - \frac{||F_n|| - 0.1}{3}\right)$$

in order to ensure that when the cattle are basically stress-free, their movement is within normal grazing speeds.

Finally, the cows' positions and velocities are adjusted by F_n.

4.1.2 Vegetation Consumption and Growth

As was previously mentioned, cattle in the simulator eat at every iteration (1 second of simulation time). The amount eaten, e, is calculated as follows.

$$e = \min\left(\frac{V}{2}, e_{\max}\right)$$

where V is the vegetation quantity for the cow's location, and $e_{\max}$ is a constant defining the maximum amount that a cow will eat in one iteration.

Vegetation grows in the simulator at a rate defined for each quality patch (patches with higher quality multipliers generally have lower regrowth rates, but this is configurable). The change in vegetation level at location (x, y) is:

$$\Delta V_{(x,y)} = \frac{4rV_{(x,y)}(255 - V_{(x,y)})}{86400 \cdot 255}$$

where r is the regrowth rate, defined as the percentage growth per day, and 86400 is the number of seconds in a day [27].

4.2 Experiments

This section presents the results of a number of experiments exploring various aspects of the system. The earlier experiments feature subsets of the total simulator functionality; what portions of the simulator were active are indicated in the experiment descriptions. Furthermore, beginning with experiment 3, the expert systems used to seed the evolutionary algorithm were changed. Both sets of seed systems are discussed here.

The first two experiments used the following two expert systems to seed the initial population.

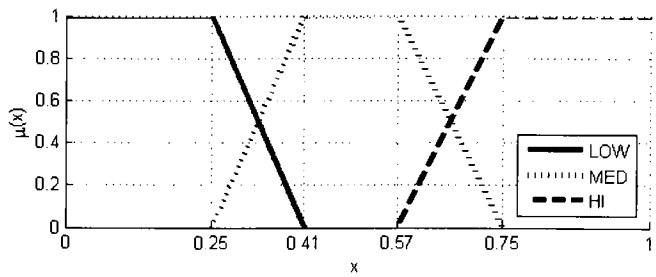


Figure 4.1: Membership functions used by all variables in seed system 1.

1 If VEGETATION is HI, then add 2.0 to score.
2 If VEGETATION is MED, then add 1.0 to score.
3 If VEGETATION is LOW, then add -1.5 to score.
4 If CELL-AGE is HI, then add 1.0 to score.
5 If CELL-AGE is LOW, then add -1.0 to score.
6 If HAPPINESS is HI, then add 2.0 to score.
7 If HAPPINESS is MED, then add 0.5 to score.
8 If HAPPINESS is LOW, then add -2.0 to score.

The membership functions for all variables were defined by the array $M = [0, 0.25, 0.41, 0.57, 0.75, 1]$ (see Figure 4.1). This system was intended to be very general; hence, it touches on both VEGETATION and HAPPINESS, as well as CELL-AGE; it contains both rewards and penalties related to each of these variables.

1 If VEGETATION is HI, then add 3.0 to score.
2 If VEGETATION is LOW, then add -1.75 to score.
3 If HAPPINESS is HI, then add 1.0 to score.
4 If HAPPINESS is LOW, then add -3.0 to score.
5 If current-cell and HERD-DISPLACEMENT is LOW, then add 4.0 to score.

The membership functions for all variables were defined by the array $M = [0, 0, 0.5, 0.5, 1, 1]$ (see Figure 4.2). This system was intended to be more minimal as compared to the first. It also introduces into the population a more complex rule designed to reward the current cell if the cattle are behaving restfully.

Beginning with experiment 4 (full simulator functionality), the seed systems were modified for a variety of reasons. First, large unreachability penalties were added to each system in order to prevent river crossings. It

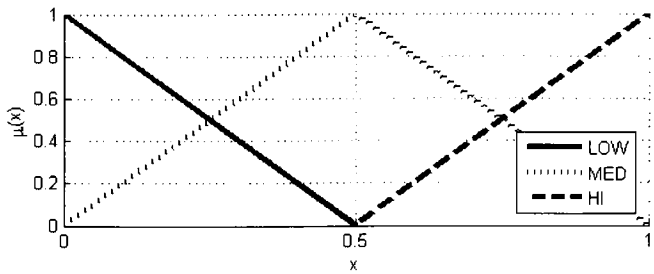


Figure 4.2: Membership functions used by all variables in seed system 2.

was decided that too much overlap existed between the first set of seeds; thus, the rules in each system were modified to make them more specialized: one focuses on coverage; the other, on happiness. The new systems were defined as follows:

1 If VEGETATION is HI, then add 2.0 to score.
2 If VEGETATION is MED, then add 1.0 to score.
3 If VEGETATION is LOW, then add -1.5 to score.
4 If CELL-AGE is HI, then add 1.0 to score.
5 If CELL-AGE is LOW, then add -1.0 to score.
6 If not reachable, then add -10.0 to score.

The membership functions for this system were not changed. The rules now concern only VEGETATION and CELL-AGE, thereby focusing exclusively on coverage. This system was assigned a θ value of 15.

1 If WATER is HI, then add 0.5 to score.
2 If WATER is MED, then add 3.0 to score.
3 If HAPPINESS is HI, then add 1.0 to score.
4 If HAPPINESS is MED, then add -0.75 to score.
5 If HAPPINESS is LOW, then add -3.0 to score.
6 If current-cell and HERD-DISPLACEMENT is LOW, then add 4.0 to score.
7 If not current-cell and HERD-DISPLACEMENT is HI, then add -2.5 to score.
8 If reachable, then add -10.0 to score.

The membership functions for WATER, HAPPINESS, and HERD-DISPLACEMENT were changed to $M = [0, 0, 0.35, 0.35, 0.6, 1]$, $M = [0, 0, 0.7, 0.7, 1, 1]$, and $M = [0, 0, 0.25, 0.45, 1, 1]$, respectively (see Figure 4.3). The new rules

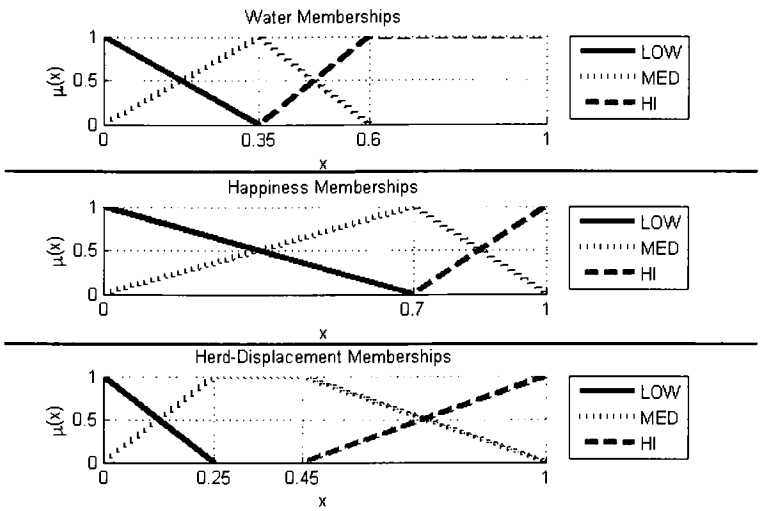


Figure 4.3: New membership functions for certain variables in seed system 4.

and memberships focus entirely on cattle stress. This system was assigned a θ value of 48.

All experiments were performed on Sun Ultra 45 workstations located in open computer laboratories in the R.I.T. Computer Science Department. In all of the evolutionary algorithm runs, each individual was simulated on its own machine; the number of machines available allowed at most two full generations (32 individuals) to be simulated in parallel. Under these conditions, running the evolutionary algorithm to convergence (generally 12–18 generations) required 1–2 weeks of runtime, depending upon load from other students' work. The five experiments discussed in this section consisted of nine evolutionary algorithm runs, some of which were run in parallel when possible. Conducting of all these experiments required approximately 63 days of runtime.

4.2.1 Experiment 1: Herd Drift

This experiment deals with the simplest possible environment: all vegetation is considered to be of the same quality, the ground is perfectly flat, there are no obstacles, and the cattle do not drink or get thirsty (thus, the simulation field contains no water). Fitness is based solely on coverage and pasture usage.

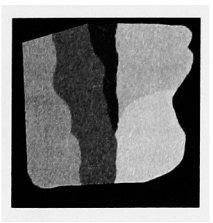


Figure 4.4: Initial state of the simulation field.

The initial environment (see Figure 4.4) consists of five regions of varying vegetation levels (represented by intensity in the grayscale image; greater intensity maps to a higher vegetation level). The surrounding black region represents a single large obstacle used to keep the cattle within the pasture.

Because there is no elevation or water in this simulation, the only forces affecting the cattle are those relating to vegetation (but not vegetation quality), and those relating to obstacle avoidance and fence stimuli.

Coverage-based (χ^2) fitness was the only fitness metric employed in this experiment. Truncation selection, rather than fitness proportional selection, was used to choose survivors.

Results

Without fences (i.e., when the cattle are given total freedom), the average fitness over five trials was $\chi^2 = 1.3884$, and the best was $\chi^2 = 0.9135$ (see Figure 4.5).

The evolutionary algorithm was able, after six generations, to produce an expert system whose fitness was $\chi^2 = 0.3167$. After nine more generations, no lower fitness was obtained, so the algorithm was considered to have converged. Figure 4.5 shows the final vegetation levels and histogram for the fittest expert system. It can be seen clearly that the coverage in this instance, although not perfect, was superior to that obtained without fences. The following final expert system was produced:

1 If VEGETATION is HI, then add 2.0 to score.
2 If VEGETATION is MED, then add 1.0 to score.
3 If VEGETATION is LOW, then add −1.5 to score.
4 If CELL-AGE is HI, then add 1.0 to score.
5 If current-cell and HERD-DISPLACEMENT is LOW, then add 4.0 to score.
6 If OBSTACLES are LOW or current-cell, then add −1.35 to score.

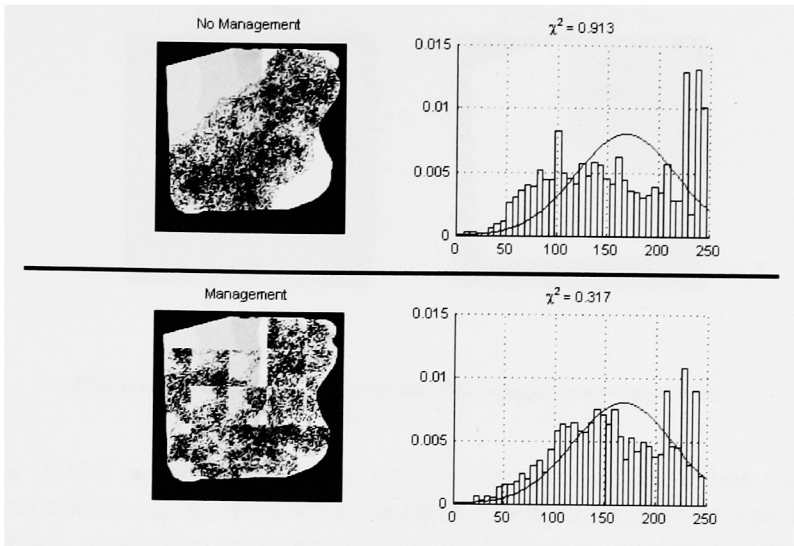


Figure 4.5: Best χ^2 fitness with management. Left: vegetation levels after 150 hours; right: final vegetation distribution histogram and ideal distribution curve.

The membership functions for all of the variables used in this system were defined by the array $M = [0, 0.25, 0.41, 0.57, 0.75, 1]$ (see Figure 4.1). The focus of this system is on visiting cells that have moderate to high vegetation levels, and preferably ones that have not been visited in awhile. The last rule is curious, and was most likely randomly generated. Since there were no obstacles in the field for this experiment, the rule reduces to a 1.35[1] penalty to the score for the current cell, overriding the inherent bonus granted to the current cell. This causes moves to occur more frequently, yielding greater coverage, but also likely increases stress to the cattle, who may not wish to be moving so often.

4.2.2 Experiment 2: Vegetation Quality and Elevation

This experiment builds upon the last one by increasing the complexity of the cattle simulation. In this set of simulations, vegetation types, with varying nutritional values and regrowth rates, are introduced. Also, elevation is made a factor. Finally, the attractor locations were moved to be on the same side of the field, in order to make the cattle less well-behaved.

[1]Randomly generated consequent values have been rounded to two decimal places in all examples.

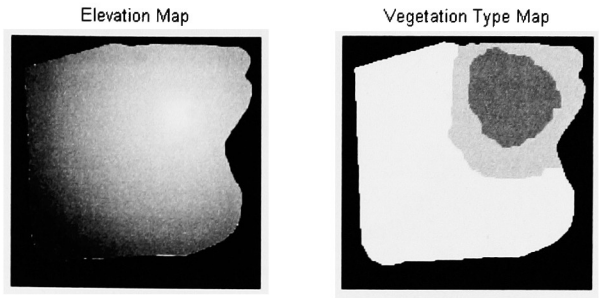


Figure 4.6: Left: field elevation; right: vegetation type map.

The initial environment is the same as in experiment 1, but with the following additions: an elevation map and a vegetation type map (see Figure 4.6). Each vegetation type has associated with it a quality factor and a regrowth rate. For this experiment, the quality factors were 1, 1.3, and 1.65, associated with the white, light gray, and dark gray regions, respectively. These regions had regrowth rates of 0.1, 0.075, and 0.05, which define the maximum growth per day.

This experiment was conducted twice: once using the coverage-based fitness function, and once using the stress-based fitness function.

Results

In this experiment, uninhibited grazing yielded an average fitness of $\chi^2 = 5.453$, and a best fitness of $\chi^2 = 3.693$, again over five trials (see Figure 4.7).

After 15 generations, the evolutionary algorithm with coverage-based

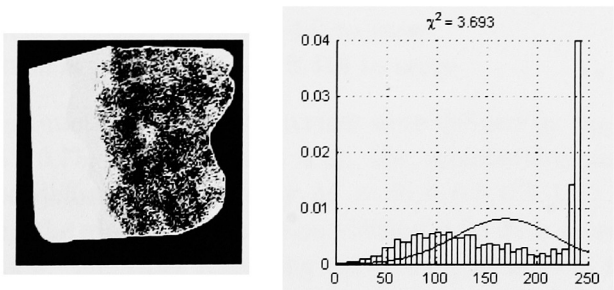


Figure 4.7: Best χ^2 fitness without management. Left: vegetation levels after 150 hours; right: final vegetation distribution histogram and ideal distribution curve.

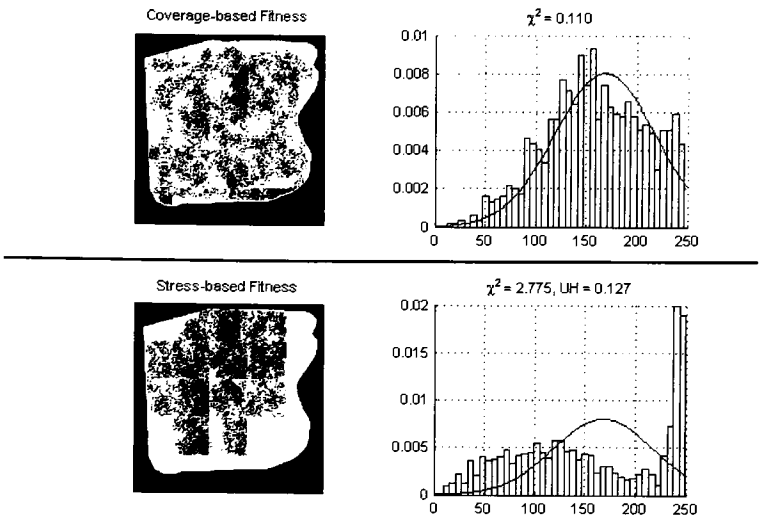


Figure 4.8: Top: best χ^2 fitness with management; bottom: best stress-based fitness. Left: vegetation levels after 150 hours; right: final vegetation distribution histogram and ideal distribution curve.

fitness yielded an expert system with a fitness of $\chi^2 = 0.110$. This value persisted for three more generations, so the algorithm was considered to have converged. Figure 4.8 shows the final vegetation levels and histogram for the fittest expert system.

The following final expert system was produced:

1 If VEGETATION is HI, then add 2.0 to score.
2 If VEGETATION is LOW, then add -1.75 to score.
3 If HAPPINESS is HI, then add -3.464 to score.
4 If HAPPINESS is MED. then add 4.0 to score.
5 If HAPPINESS is LOW, then add 0.414 to score.

The membership functions for VEGETATION were defined by the array $M = [0, 0.25, 0.41, 0.57, 0.75, 1]$ (see Figure 4.1); the membership functions for HAPPINESS were defined by the array $M = [0, 0, 0.5, 0.5, 1, 1]$ (see Figure 4.2). Once again, the system's focus is on visiting cells with more (or higher quality) vegetation. The rules regarding cattle happiness seem to be aimed at making the cattle miserable; however, it was determined during this experiment that the cow happiness estimate calculation used by the expert system was never yielding values lower than 0.5 out of 1. Thus, the first

happiness rule was almost always in effect, causing a blanket reduction in all scores (note that this was remedied before switching to the stress-based fitness metric).

The evolutionary algorithm converged in 5 generations using the stress-based fitness metric at a fitness of 0.127 (i.e., an average of 1 shock every 7–8 seconds). The coverage metric for the final expert system was $\chi^2 = 2.775$ (see Figure 4.8). The stress-based metric produced an expert system with the following rules set:

1 If VEGETATION is HI, then add 2.0 to score.
2 If VEGETATION is MED, then add 1.0 to score.
3 If VEGETATION is LOW, then add −1.5 to score.
4 If CELL-AGE is HI, then add 1.0 to score.
5 If CELL-AGE is LOW, then add −1.0 to score.
6 If HAPPINESS is HI, then add 2.0 to score.
7 If HAPPINESS is MED, then add 0.5 to score.
8 If HAPPINESS is LOW, then add −2.0 to score.

The membership functions for all variables used in the rule set were defined by the array $M = [0, 0.25, 0.41, 0.57, 0.75, 1]$ (see Figure 4.1). This is one of the seed systems; it assigns nearly equal weight to HAPPINESS and VEGETATION, with LOW HAPPINESS being punished slightly more harshly than LOW VEGETATION, but with HI values for either being rewarded identically.

4.2.3 Experiment 3: Full Simulator Functionality

The first two experiments were proof-of-concept trials intended to demonstrate the approach's readiness to operate with full simulator functionality. Beginning with this experiment, the grazing management algorithm was pitted against the full complexity level offered by the simulation; specifically, water sources were added to the field along with clumps of trees, and a new, more realistic elevation map was developed. Figure 4.9 depicts the field layout and elevation.

The management task is significantly more difficult with the addition of water sources and the related simulated cattle behavior. A majority of the management algorithm's cells do not contain a water source, so the system must learn to rotate the cattle among these cells, and then return them to a cell containing water when the cattle become thirsty.

Beginning with this experiment, the evolutionary algorithm was prevented from generating rules whose antecedents consisted only of the *cur-*

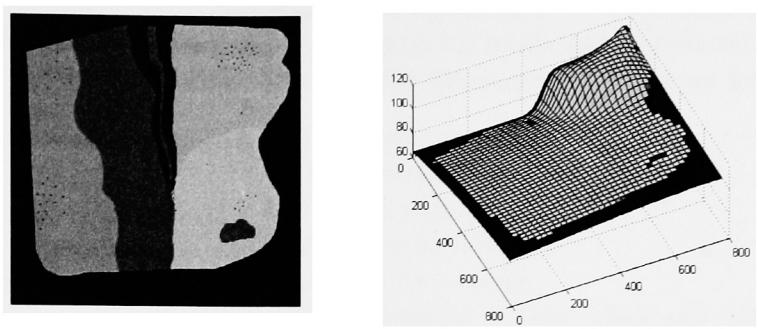


Figure 4.9: Left: Aerial view of the final simulation field incarnation; right: 3-D view illustrating elevation changes throughout the field.

rent-cell operator. Allowing rules of this form to be generated significantly reduces the likelihood of producing complex rules.

As in the previous experiment, this one was conducted twice in order to compare the results of coverage-based and stress-based fitness. Additionally, this experiment was performed using the combined fitness metric, with the expectation that this metric would outperform the separate ones.

Results

Continuous grazing on the new field yielded an average fitness of $\chi^2 = 6.887$ over five trials, and a best fitness of $\chi^2 = 5.849$. Because the combined fitness is calculated as $f = \chi^2 e^u$, and $u = 0$ when the cattle are allowed to roam free, these values are also the mean and best combined fitness metrics. Figure 4.10 depicts the final state of the field in the best case.

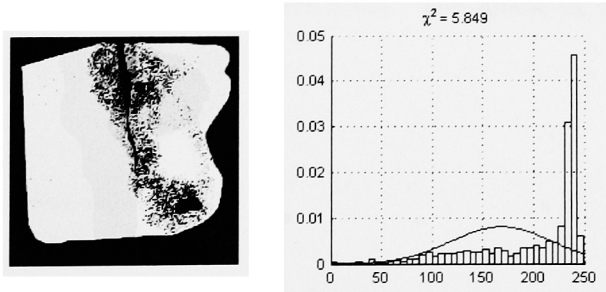


Figure 4.10: Best χ^2/combined fitness without management. Left: vegetation levels after 150 hours; right: final vegetation distribution histogram and ideal distribution curve.

Coverage-based fitness converged on a fitness of $\chi^2 = 3.855$ after 8 generations. Figure 4.11 shows the final vegetation levels and distribution histogram for the winning expert system. The fittest system's rule set follows.

1 If WATER is HI, then add 0.5 to score.
2 If WATER is MED, then add 3.0 to score.
3 If HAPPINESS is HI, then add 1.0 to score.
4 If HAPPINESS is MED, then add -3.0 to score.
5 If HAPPINESS is LOW, then add -0.75 to score.
6 If current-cell and HERD-DISPLACEMENT is LOW, then add 4.0 to score.
7 If not current-cell and HERD-DISPLACEMENT is HI, then add -2.5 to score.
8 If not reachable, then add -10.0 to score.
9 If current-cell, then add -1.01 to score.

The membership functions for the variables WATER, HAPPINESS, and HERD-DISPLACEMENT were defined by the arrays $M = [0, 0, 0.35, 0.35, 0.6, 1]$, $M = [0, 0, 0.7, 0.7, 1, 1]$, and $M = [0, 0, 0.25, 0.45, 1, 1]$, respectively (see Figure 4.3). Interestingly enough, this system does not directly concern itself with vegetation levels; rather, through concern for cattle restlessness and a reduction in the implicit current cell bonus (note that after this experiment, this behavior was disallowed), the expert system keeps the cattle moving between cells, thereby avoiding overgrazing. The large negative score modification resulting from "not reachable" prevents the system from pushing the cattle through the river (this can be done, but the cattle are extremely resistant).

After 9 generations, stress-based fitness converged to a fitness of 0.013 (an average of approximately 1 shock per 77 seconds). This yielded a coverage metric of $\chi^2 = 24.836$ (see Figure 4.11). The following rule set was produced:

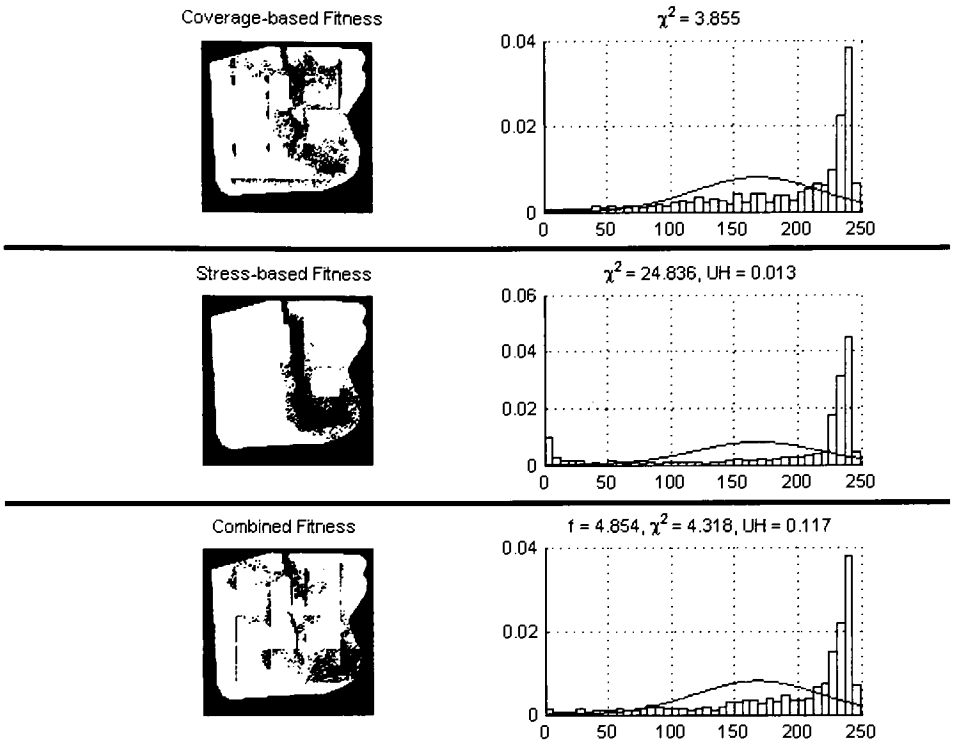


Figure 4.11: Top: best χ^2 fitness with management; middle: best stress-based fitness; bottom: best combined fitness. Left: vegetation levels after 150 hours; right: final vegetation distribution histogram and ideal distribution curve.

1 If WATER is HI, then add 0.5 to score.
2 If WATER is MED, then add 3.0 to score.
3 If HAPPINESS is HI, then add 1.0 to score.
4 If CELL-AGE is LOW, then add -1.0 to score.
5 If HAPPINESS is LOW, then add -3.0 to score.
6 If not current-cell and HERD-DISPLACEMENT is HI, then add -2.5 to score.
7 If not reachable, then add -10.0 to score.
8 If current-cell, then add -3.8 to score.
9 If HAPPINESS is MED, then add 2.97 to score.
10 If EVENNESS is MED, then add -0.02 to score.

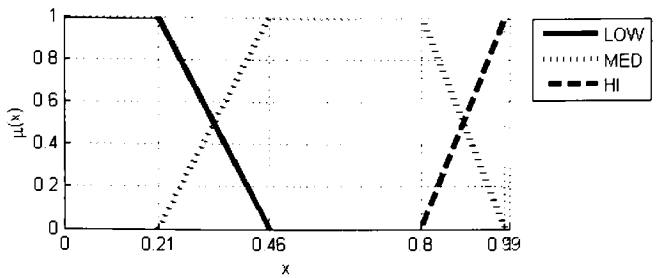


Figure 4.12: Functions used by the stress-based fitness winner to determine fuzzy memberships of EVENNESS.

The membership functions for the variables WATER, HAPPINESS, and HERD-DISPLACEMENT were defined as in seed system 4 (see Figure 4.3). The membership functions for CELL-AGE were defined as in seed system 1 (see Figure 4.1). The membership functions for EVENNESS were defined by the array $M = [0, 0.21, 0.46, 0.8, 0.99, 1]$ (see Figure 4.12); these values were generated through mutation. This rule set is very similar to the one generated via coverage-based fitness, but with two significant difference. First, rather than a large penalty for MED HAPPINESS, there is a large bonus; second, the blanket reduction to the current cell bonus is so large that the current cell suffers a large penalty (the implicit bonus is 1). So, the system rarely remains in the same cell; however, because of the significant bonuses to MED WATER and MED HAPPINESS, the system will cycle back and forth between the same few cells containing moderate amounts of water. This is illustrated well in the image of the final vegetation levels: vast portions of the field are untouched, while areas around the pond and river are extremely overgrazed.

Finally, the combined fitness function converged in 15 generations, yielding an expert system with a combined fitness of 4.854. The coverage metric for this system was $\chi^2 = 4.318$; the stress-based metric was 0.117, or an average of approximately 1 shock every 8–9 seconds (see Figure 4.11). The evolved rule set is shown below.

1 If HAPPINESS is HI, then add 2.0 to score.
2 If WATER is MED, then add 3.0 to score.
3 If VEGETATION is HI, then add -0.75 to score.
4 If CELL-AGE is LOW, then add -1.0 to score.
5 If HAPPINESS is LOW, then add -3.0 to score.
6 If current-cell and HERD-DISPLACEMENT is LOW, then add -3.14 to score.
7 If not reachable, then add -10.0 to score.

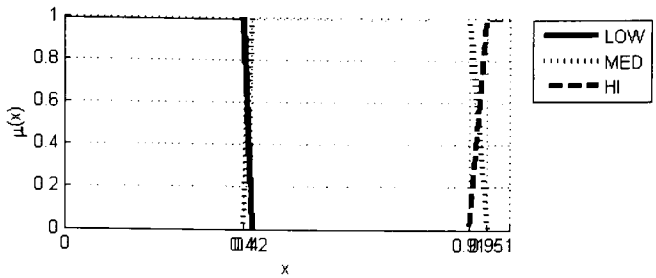


Figure 4.13: Membership functions used to determine the fuzzy membership of HAPPINESS in the combined-fitness winning expert system.

The membership functions for HERD-DISPLACEMENT were defined as in seed system 4 (see Figure 4.3). The membership functions for HAPPINESS were defined by the array $M = [0, 0.4, 0.42, 0.91, 0.95, 1]$ (see Figure 4.13). The membership functions for the remaining variables used in the rules were defined by the array used in seed system 1 (see Figure 4.1). The evolutionary algorithm produced a combination of the previous two rules sets, but with some superfluous rules stripped out. Thus, for each of WATER, VEGETATION, and CELL-AGE, only one rule is present. This system suffers two flaws: first, that it assigns a slight penalty for HI VEGETATION; second, that it waits too long to respond to certain behavior — LOW HAPPINESS and LOW HERD-DISPLACEMENT — that can be indicative of increased cattle stress. Reacting to LOW HERD-DISPLACEMENT is difficult as this condition could indicate that the cattle are grazing happily, or it could indicate that they are clustered against a fence, trying to escape. This system takes it to mean the latter; however, at this point the cattle should probably have already been moved.

The results of this experiment demonstrate the necessity of the combined fitness function; stress-based fitness, especially, is not effective on its own, yielding an atrocious χ^2 score. Furthermore, combined fitness is shown here to successfully improve coverage over continuous grazing without cause high stress levels among the cattle.

4.2.4 Experiment 4: On-line Adaptation of the Expert System

This set of experiments features the introduction of the on-line adaptation portion of the grazing management algorithm. The cattle simulation reached its full complexity previously; no new functionality was added during these

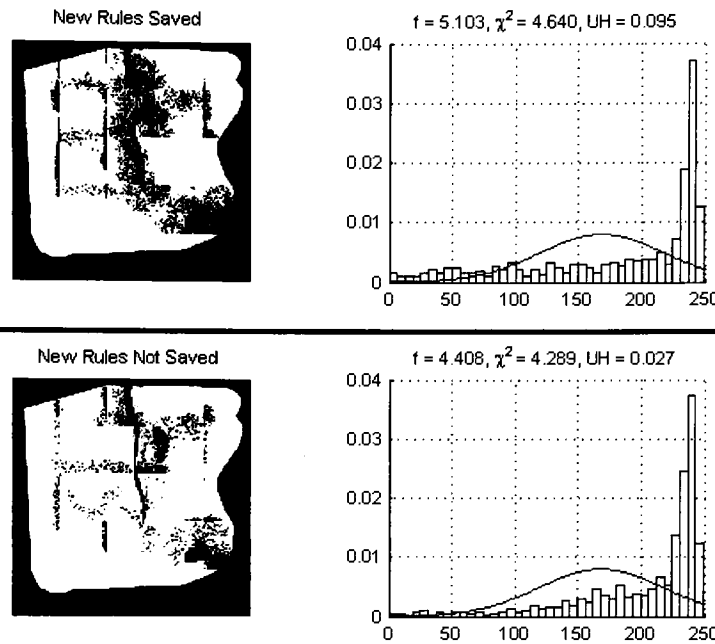


Figure 4.14: Top: best fitness with adapted rules saved; bottom: best fitness with adapted rules discarded. Left: vegetation levels after 150 hours; right: final vegetation distribution histogram and ideal distribution curve.

tests. Fitness was calculated using the combined fitness metric.

Two methods for handling the adapted rules with respect to the evolutionary algorithm were explored. In the first case, at the completion of a simulation, the adapted versions of the rule sets were stored, so that the evolutionary algorithm then operated on these new rules; here the intention is to pass what is learned on to later generations. In the second case, the new rules were not saved; rather than passing along what is learned, the intent here is to evolve systems most capable of adapting to the situation at hand.

Results

The previous section's baseline continuous grazing results are also used for comparison here (see Figure 4.10). Recall that the baseline best combined fitness was 5.849.

In the case wherein adapted rules were saved before performing crossover and mutation, convergence took 14 generations, yielding a best fitness of 5.103 (see Figure 4.14); coverage-based fitness was $\chi^2 = 4.640$, and stress-based fitness was 0.095 (average 10–11 shocks per second). The winning rule set follows.

1 If WATER is HI, then add 0.5 to score.
2 If WATER is MED, then add 3.0 to score.
3 If HAPPINESS is HI, then add −0.99 to score.
4 If CELL-AGE is HI, then add 1.31 to score.
5 If HAPPINESS is LOW, then add −3.0 to score.
6 If HERD-DISPLACEMENT is HI, then add 0.46 to score.

The membership functions for HAPPINESS were defined as in seed system 4 (see Figure 4.3); the functions for the remaining variables were defined as in seed system 1 (see Figure 4.1).

The system's θ was 53, meaning that in order to respond to an observed cause-effect relationship, that relationship would have to be seen 54 times; with the system running once every 5 minutes, that equates to 4.5 hours. Adaptation in this system is likely quite rare.

Without saving adapted rules, a best fitness of 4.408 was obtained in 7 generations. This system yielded a coverage-based fitness of $\chi^2 = 4.289$, and a stress-based fitness of 0.027 (average 1 shock every 37 seconds). Figure 4.14 depicts the winning system's final field state and vegetation distribution. The resulting rules:

1 If WATER is HI, then add 0.5 to score.
2 If WATER is MED, then add 3.0 to score.
3 If HAPPINESS is HI, then add 1.0 to score.
4 If HAPPINESS is MED, then add −0.75 to score.
5 If HAPPINESS is LOW, then add −3.0 to score.
6 If current-cell and HERD-DISPLACEMENT is LOW, then add 4.0 to score.
7 If not current-cell and HERD-DISPLACEMENT is HI, then add −2.5 to score.
8 If not reachable, then add 0.79 to score.

The membership functions for HAPPINESS and HERD-DISPLACEMENT were defined as in seed system 4 (See Figure 4.3); the functions for WATER were defined as in seed system 1 (see Figure 4.1). Rule 8 is very curious: without a large penalty for unreachability, the system could very likely attempt to push the cattle through the river. In the simulator, water is considered to be an

obstacle, so the cattle receive large repulsive forces if they attempt to enter it. If the system attempts to push them into it, then they will receive large repulsive forces from the virtual fence if they do *not* enter the river. The result is the simultaneous application of a large force directed away from the river (and thus, opposite from the cow's heading), and a large force directed perpendicular to the cow's heading. The result would be that the cow spins in place (see Section 4.1.1). Ultimately, the cattle will be forced across the river (this has been witnessed in simulation); however, the entire herd is constantly shocked throughout the process. Therefore, it may be concluded from the low stress-based fitness that this system either avoided crossing the river in another manner, or that it was so successful in maintaining cattle happiness otherwise, that it could force the cows to cross the river while still maintaining a low average stress level. Visual inspection of the final field state indicates that the cattle were forced across the river at least once, and possibly twice (see Figure 4.15). Therefore, had a penalty been in place for unreachability, this system may have been extremely successful.

In stark contrast to the previous system's $\theta = 53$, this system's θ was 4, allowing learning to occur as often as once every 20 minutes simulation time. The behavior of this system, therefore, most likely changes a great deal during simulation. Evidence of this can once again be seen in the final field state. There are definite regions of overgrazing on the right side of the field near the herd's starting location. On the left side of the river, overgrazing occurs much less frequently, indicating that by the time the cattle had been moved to that side, on-line adaptation had taught the system to better avoid this undesirable behavior (see Figure 4.15).

4.2.5 Experiment 4: Reduced Thirst Frequency

This experiment was motivated by the following observation: the coverage-based fitness when the cattle do not periodically require access to water was 0.110; the best combined fitness obtained once thirst was added was 4.408. In order to aid in determining what is necessary to bridge this rather significant gap, an experiment was conducted in which the parameter defining how often the cattle will optimally drink was set to 4 hours (it was previously 2 hours), providing a partial alleviation of the challenge of maintaining proper coverage while returning to water at regular intervals.

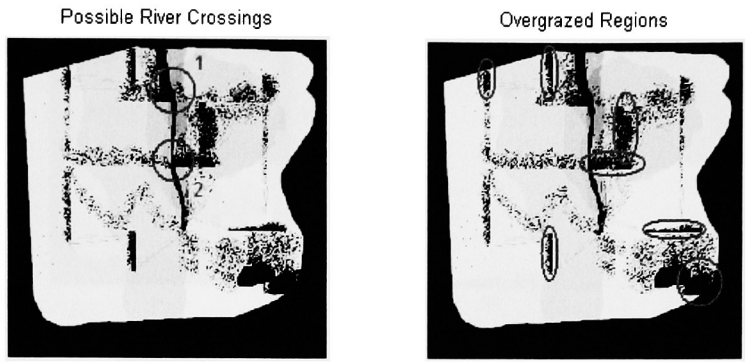


Figure 4.15: Left: locations of possible river crossings; given that the only trail leading to the northwest river bank crosses the river, this crossing almost certainly occurred. Crossing 2 is quite likely, but not certain. Right: frequency of overgrazing on the left and right sides of the field.

Results

A new set of continuous grazing control results wherein the adjusted drinking rate was used were obtained for this experiment. The mean fitness over five simulations was 6.165, with a best fitness of 5.925 (see Figure 4.16).

The evolutionary algorithm converged after 10 generations to a combined fitness of 4.258. Coverage-based fitness was 3.930, and stress-based fitness was 0.080 (average 1 shock every 12–13 seconds). The following expert system was produced:

1 If VEGETATION is LOW, then add −1.5 to score.
2 If CELL-AGE is HI, then add 1.0 to score.
3 If HAPPINESS is MED, then add −1.5 to score.
4 If CELL-AGE is LOW, then add −1.0 to score.
5 If not reachable, then add −10.0 to score.
6 If LINEARITY is not LOW, then add −0.85 to score.
7 If VEGETATION is MED and CELL-AGE is LOW, then add −3.51 to score.

The membership functions for VEGETATION and CELL-AGE were defined as in seed system 2 (see Figure 4.2); the functions for HAPPINESS and LINEARITY were defined as in seed system 1 (see Figure 4.1). Rule 7 excellently summarizes a large part of management-intensive grazing; it says, basically, that if the cattle have eaten a reasonable amount of vegetation, and they've

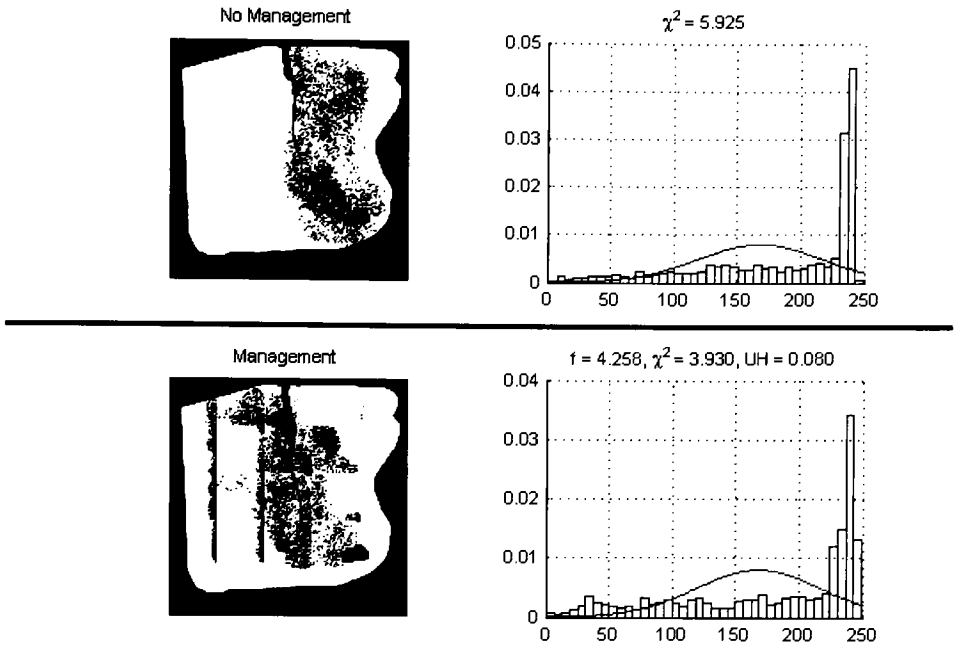


Figure 4.16: Top: best continuous grazing fitness; bottom: best fitness with management. Left: final field state; right: vegetation distribution and ideal distribution curve.

been in one place for awhile, then move them (or do not send them back, as the case may be). Most of the system, in fact, is comprised of reasons to leave a cell, or to allow a cell to lie fallow. Rule 6 is curious: penalizing the linear movement of the cattle seems counter-intuitive; however, it may be that the simulation field does not always lend itself to linear grazing. That none of the previous systems even contained a LINEARITY rule would lend credence to this theory.

This expert system had $\theta = 63$, meaning that on-line adaptation occurred at most once every 5.25 hours (in fact, it was likely far less common than this). Had this system been learning more often, it quite likely would have been more successful.

4.3 Results Analysis

The experimental results are quite promising. In all experiments, the winning expert system allowed the management algorithm to yield a better

fitness than continuous grazing (with the exception of stress-based fitness in experiment 3; however, this fitness metric was employed to determine how the final system would differ from that produced by a coverage-based fitness function; thus, it was not expected to outperform continuous grazing in terms of field coverage). The successes of the combined fitness metric demonstrate an ability to mitigate the coverage-stress trade-off, which is the heart of the grazing management problem.

Furthermore, the on-line adaptation algorithm has been shown to increase system performance when the adapted rules are not saved. Saving the adapted rules carries with it an inherent risk: if the adapted rules are good, then saving them is useful, as they will become available in the genetic pool; however, if the adapted rules are poor, then the genetic pool becomes poisoned by these rules. That the system evolved through saving adapted rules had a θ value of 53, in effect forcing minimal on-line learning, is evidence of this suspected behavior; the evolutionary algorithm produces systems that rarely adapt in an attempt to prevent the poisoning of the genetic pool.

Storing the original rule sets is beneficial in other ways, as well. The purpose of on-line adaption is to provide a generic grazing management system with the means to customize itself to current animal and field behaviors and conditions. Thus, it is desirable to persist the generic rules between generations in order to produce general solutions rather than solutions adapted specifically to the field or fields used in training.

Although the results indicate a technical success, they are far from optimal. Most notably, the significant gap between fitness attained before the addition of thirst to the cow model (0.110) and after (4.408) still remains. Although the system that attained the latter fitness outperformed continuous grazing, the χ^2 component of that fitness (4.289) indicates a substantial mismatch between the actual and desired vegetation distributions. Visual inspection of the final field state for this system (Figure 4.14, lower portion) concurs with this assessment: large portions of the field appear unvisited; the vegetation distribution histogram appears to fit an exponential curve far better than a normal one.

Why has no system capable of attaining truly satisfactory results in the face of the full simulation complexity been produced? The expert systems are capable of representing complex reasoning in their rules and membership functions; the on-line adaptation algorithm has been shown to effectively improve performance. The most likely limiting factor, therefore, is the evolutionary algorithm itself.

If an evolutionary algorithm's population is too small, the likelihood of

experiencing premature convergence increases because the genetic pool is too small; this is, in effect, the same problem that arises through selecting parents by truncation. The population size used in the experiments described here, 16, is quite small, and was dictated by time and equipment restraints. A larger population, coupled with a larger number of systems used for seeding the initial population, would lead to better coverage of the solution space. Additionally, a variable mutation rate that starts high and decays over time would improve the exploration of the solution space, and especially the rule space. Finally, in order to accommodate the larger population and initially high mutation rate, many more generations would be required.

Unfortunately, it was not possible to conduct experiments testing these proposed evolutionary algorithm modifications. As indicated previously, the experiments presented here required just over two months of runtime to complete; this figure does not include time spent incrementally developing the system, analyzing experiment results, or fixing errors that would surface after a week of simulations. Conducting more experiments on a much larger scale is beyond the scope of this work.

Chapter 5

Conclusion

This first foray into the field of automated grazing management using dynamic virtual fences has proven to be a successful one. The proposed management algorithm was shown to consistently improve coverage over the traditional continuous grazing method. Furthermore, both the procedure for evolving expert systems and the algorithm for adapting expert systems in an on-line manner designed and developed for this work proved successful: the evolved expert systems yielded superior grazing management results; the addition of the adaptation algorithm improved these results.

There are a number of ways, however, in which this work could be expanded and improved. A number of possible improvements to the evolutionary algorithm were discussed in the previous chapter. In addition to these measures, other avenues should also be explored. Most importantly, the use of a more complex and accurate cow model would greatly increase the realism of the simulation used to determine fitness, leading to the development of management systems more prepared for deployment on real cattle. In order to ensure that the systems being generated are truly general, multiple simulation fields should be used in fitness calculation; these should either be selected at random, or a fitness for each field should be determined, and then averaged to obtain the overall fitness of a system.

This work has demonstrated that automated grazing management is possible and attainable. Hopefully, further work will show it to be practical and successful.

References

[1] T. E. Poole, "Grazing management." University of Maryland Cooperative Extension Fact Sheet 786.

[2] C. E. Owensby, "Efficient grazing systems for rangelands," in *Ranch Management Field Day: Profitable Grass and Cattle Management. Proceedings*, pp. 13–19, 1987.

[3] N. R. Harris, D. E. Johnson, M. R. George, and N. K. McDougald, "The effect of topography, vegetation, and weather on cattle distribution at the san joaquin experimental range, california," in *Oak Woodlands: Oaks in California's Challenging Landscape. Proceedings of the Fifth Symposium on*, pp. 53–63, 2002.

[4] L. W. Turner, C. W. Absher, and J. K. Evans, "Planning fencing systems for intensive grazing management." University of Kentucky College of Agriculture Online Publication ID-74.

[5] G. D. Hanson, "Adoption of intensive grazing systems," *Journal of Extension*, vol. 33, no. 4, 1995.

[6] "Handbook of texas online." s.v. "King Ranch" http://www.tshaonline.org/handbook/online/articles/KK/apk1.html (accessed April 13, 2008).

[7] R. Meyer and T. Olsen, "Estimated costs for livestock fencing." Iowa State University Extension File B1-75, 2005.

[8] S. Barnhart, M. Duffy, and D. Smith, "Estimated costs of pasture and hay production." Iowa State University Extension File Economics 1-8 and Agronomy 3-3, 2006.

[9] Z. Butler, P. Corke, R. Peterson, and D. Rus, "Dynamic virtual fences for controlling cows," in *Experimental Robotics, 2004. 9^{th} International Conference on*, 2004.

[10] D. M. Anderson, B. Nolen, E. Fredrickson, K. Havstad, C. Hale, and P. Nayak, "Representing spatially explicit directional virtual fencing (DVF^{TM}) data," in *24th Annual ESRI International User Conference. Proceedings of the*, 2004.

[11] G. J. Bishop-Hurley, D. L. Swain, D. M. Anderson, P. Sikka, C. Crossman, and P. Corke, "Virtual fencing applications: Implementing and testing an automated cattle control system," *Computer and Electronics in Agriculture*, vol. 56, no. 1, pp. 14–22, 2007..

[12] R. Vaghan, N. Sumpter, A. Frost, and S. Cameron, "Robot sheepdog project achieves automatic flock control," in *Simulation of Adaptive Behavior on From Animals to Animats 5. Proceedings Fifth International Conference on*, (Cambridge, MA, USA), pp. 489–493, MIT Press, 1998.

[13] R. Vaughan, N. Sumpter, A. Frost, and S. Cameron, "Robot control of animal flocks," in *Proceedings of the 1998 IEEE ISIC/CIRA/ISAS Joint Conference*, pp. 277–282, 1998.

[14] Z. Butler, "Corridor planning for natural agents," in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pp. 499–504, 2006.

[15] C. Perneel, J.-M. Themlin, J.-M. Renders, and M. Acheroy, "Optimization of fuzzy expert systems using genetic algorithms and neural networks," *Fuzzy Systems, IEEE Transactions on*, vol. 3, no. 3, pp. 300–312, 1995.

[16] J. Drabarek, R. Wirski, and W. Madej, "Genetic algorithms applied to hybrid expert systems," in *Electronics, Circuits and Systems, 2002. 9th International Conference on*, vol. 3, pp. 1263–1266, 2002.

[17] J. F. M. Amaral, R. Tanscheit, M. A. C. Pacheco, and M. M. R. Vellasco, "Evolutionary fuzzy system design and implementation," in *Neural Information Processing, 2002. ICONIP '02. Proceedings of the 9th International Conference on*, vol. 4, pp. 1872–1876, 2002.

[18] M.-R. Akbarzadeh-T, I. Mosavat, and S. Abbasi, "Friendship modeling for cooperative co-evolutionary fuzzy systems: A hybrid ga-gp algorithm," in *Fuzzy Information Processing Society, 2003. NAFIPS 2003. 22nd International Conference of the North American*, pp. 61–66, 2003.

[19] R. Calvo, M. Figueiredo, and E. A. Antonelo, "Evolutionary fuzzy system for architecture control in a constructive neural network," in *Computational Intelligence in Robotics and Automation, 2005. CIRA 2005. Proceedings. 2005 IEEE International Symposium on*, pp. 541–546, 2005.

[20] H. Y. Xu and G. Vukovich, "Fuzzy evolutionary algorithms and automatic robot trajectory generation," in *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence. Proceedings of the First IEEE Conference on*, vol. 2, pp. 595–600, 1994.

[21] Y. Shi, R. Eberhart, and Y. Chen, "Implementation of evolutionary fuzzy systems," *Fuzzy Systems, IEEE Transactions on*, vol. 7, no. 2, pp. 109–119, 1999.

[22] J.-S. R. Jang, "ANFIS: Adaptive-network-based fuzzy inference system," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 23, no. 3, pp. 665–685, 1993.

[23] R. Bogacz and C. Giraud-Carrier, "Learning meta-rules of selection in expert systems," in *Expert Systems, Proceedings. Fourth World Conference on*, pp. 576–581, 1998.

[24] L. Cinque, M. Crispoldi, and S. Levialdi, "Use of meta-knowledge in an expert system: a real case application," in *Industrial Applications of Machine Intelligence and Vision. International Workshop on*, pp. 176–180, 1989.

[25] B. Jankowska, "Truth maintenance in an expert system with uncertainty," *Schedae Informaticae*, vol. 14, pp. 61–74, 2005.

[26] A. Pirzadeh and W. Snyder, "A unified solution to coverage and search in explored and unexplored terrains using indirect control," in *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*, vol. 3, pp. 2113–2119, 1990.

[27] "Planting grass — how fast does it grow?." http://floridaturf.com/bermuda/compound.htm (accessed May 13, 2008).